

GETTING STARTED WITH BATCH (TOPS-20)

Order Number: AA-C781B-TM

August 1978

This document describes to the reader how to get started with the TOPS-20 GALAXY Batch System.

This document supersedes the document Getting Started With Batch, Order Number DEC-20-OBGSA-A-D, and its update DEC-20-OBGSA-A-DN1.

OPERATING SYSTEM: TOPS-20 Version 3A

SOFTWARE VERSION: GALAXY Version 3

To order additional copies of this document, contact the Software Distribution Center, Digital Equipment Corporation, Maynard, Massachusetts 01754

digital equipment corporation • maynard, massachusetts

First Printing, November 1976
Revised: January 1978
Revised: August 1978

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by DIGITAL or its affiliated companies.

Copyright © 1976, 1978 by Digital Equipment Corporation

The postage-prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DIGITAL	DECsystem-10	MASSBUS
DEC	DECTape	OMNIBUS
PDP	DIBOL	OS/8
DECUS	EDUSYSTEM	PHA
UNIBUS	FLIP CHIP	RSTS
COMPUTER LABS	FOCAL	RSX
COMTEX	INDAC	TYPESET-8
DDT	LAB-8	TYPESET-11
DECCOMM	DECSYSTEM-20	TMS-11
ASSIST-11	RTS-8	ITPS-10

CONTENTS

	Page
PREFACE	v
REFERENCES	vi
CONVENTIONS USED IN THIS MANUAL	vii
SYMBOLS USED IN THIS MANUAL	viii
GLOSSARY	Glossary-1
CHAPTER 1 INTRODUCTION	1-1
1.1 WHAT BATCH IS	1-1
1.2 HOW TO USE BATCH	1-2
1.2.1 Running Your Job	1-2
1.2.2 Receiving Your Output	1-3
1.2.3 Recovering from Errors	1-3
1.3 SUMMARY	1-3
CHAPTER 2 ENTERING A BATCH JOB FROM A TERMINAL	2-1
2.1 CREATING THE CONTROL FILE	2-2
2.1.1 Format of Lines in the Control File	2-3
2.2 SUBMITTING THE JOB TO BATCH	2-4
2.2.1 Switches	2-5
2.2.2 Examples of Submitting Jobs	2-6
2.3 BATCH COMMANDS	2-8
2.3.1 The @IF Command	2-8
2.3.2 The @ERROR Command	2-9
2.3.3 The @NOERROR Command	2-9
2.3.4 The @GOTO Command	2-10
2.3.5 The @BACKTO Command	2-11
2.4 SPECIFYING ERROR RECOVERY IN THE CONTROL FILE	2-12
CHAPTER 3 ENTERING A BATCH JOB FROM CARDS	3-1
3.1 BATCH CONTROL CARD FORMAT CONVENTIONS	3-2
3.2 BATCH CONTROL CARD COMMANDS	3-2
3.2.1 Starting a Job - The \$JOB Card	3-2
3.2.2 Identifying Yourself - The \$PASSWORD Card	3-5
3.2.3 Ending a Job - The \$EOJ Card	3-6
3.2.4 Creating a File - The \$CREATE Card	3-6
3.2.5 Compiling a Program - The \$-language Card	3-7
3.2.6 Executing a Program - The \$EXECUTE Card	3-10
3.2.7 Executing a Program with Data - The \$DATA Card	3-10
3.2.7.1 Reading from a Spooled Card-Reader File	3-12
3.2.7.2 Naming Data Files on the \$DATA Card	3-14

CONTENTS (CONT.)

	Page
3.2.8 End of Data Input - The \$EOD Card	3-16
3.2.9 System Commands - The \$TOPS20 Card	3-16
3.2.10 Error Recovery - The \$ERROR and \$NOERROR Cards	3-18
3.3 SETTING UP YOUR CARD DECK	3-20
3.4 PUTTING COMMANDS INTO THE CONTROL FILE FROM CARDS	3-21
3.4.1 Card Decks for Programs that Do Not Have Special Control Cards	3-22
3.5 SPECIFYING ERROR RECOVERY IN THE CONTROL FILE	3-26
 CHAPTER 4 INTERPRETING YOUR PRINTED OUTPUT	 4-1
4.1 OUTPUT FROM YOUR JOB	4-1
4.2 BATCH OUTPUT	4-1
4.3 OTHER PRINTED OUTPUT	4-2
4.4 SAMPLE BATCH OUTPUT	4-2
4.4.1 Sample Output of a Job from a Terminal	4-2
4.4.2 Sample Output of a Job on Cards	4-5
 CHAPTER 5 EXAMPLES OF COMMON TASKS WITH BATCH	 5-1
5.1 USING THE TERMINAL TO ENTER JOBS	5-1
5.2 USING CARDS TO ENTER JOBS	5-11
 INDEX	 Index-1

FIGURES

	Page
FIGURE 3-1 Batch Card Deck Using TOPS-20 Commands	3-18
3-2 Typical Program Card Deck	3-20
3-3 Use of Control Cards to Compare Two Card Decks	3-22
3-4 BASIC Program Card Deck with Integral Data	3-24
3-5 BASIC Program Card Deck with Provisions for Terminal Data Input	3-25
3-6 Card Deck with Error Statement	3-26
3-7 Card Deck with Error Recovery Program	3-28
3-8 Card Deck Using GOTO Statement	3-30
4-1 COBOL Print Program Card Deck	4-5
5-1 ALGOL Job Entry Card Deck	5-11
5-2 BASIC Job Entry and Run Card Deck	5-14
5-3 FORTRAN Card Deck That Prevents Execution on Error	5-17
5-4 COBOL Program Card Deck Using Data From Magnetic Tape	5-21

PREFACE

Getting Started With Batch (TOPS-20) has been written for you, if you have a rudimentary knowledge of Batch processing or are familiar with at least one of the following:

1. a programming language
2. the timesharing services of the DECSYSTEM-20
3. card processing on other systems

HOW TO USE THIS MANUAL

If you input your jobs through interactive terminals, the following chapters are recommended:

- | | |
|-----------|---|
| Chapter 1 | Introduction |
| Chapter 2 | Entering a Batch Job from a Terminal |
| Chapter 4 | Interpreting Your Printed Output, Section 4.4.1 |
| Chapter 5 | Using the Terminal to Enter Jobs, Section 5.1 |

If you input your jobs from cards, the following chapters are recommended:

- | | |
|-----------|---|
| Chapter 1 | Introduction |
| Chapter 3 | Entering a Batch Job from Cards |
| Chapter 4 | Interpreting Your Printed Output, Section 4.4.2 |
| Chapter 5 | Using Cards to Enter Jobs, Section 5.2 |

REFERENCES

Not all of the commands and card formats for Batch are described in this manual. If you want to know more about Batch you can refer to the DECSYSTEM-20 (TOPS-20) Batch Reference Manual. In addition, all components of Batch processing are referred to as Batch in this manual. For a complete description of these components, refer to the DECSYSTEM (TOPS-20) Batch Reference Manual.

An elementary description of the basic TOPS-20 system commands can be found in the document Getting Started with DECSYSTEM-20 (TOPS-20). The DECSYSTEM-20 (TOPS-20) User's Guide contains additional descriptions of the TOPS-20 commands available to you.

Error messages that occur while Batch is processing but which are not defined in this manual are explained in applicable system manuals. For example, if your FORTRAN program fails to compile successfully, the error message you receive from the FORTRAN compiler can be found in the DECSYSTEM-20 (TOPS-20) FORTRAN Reference Manual. For errors that may occur in a Batch process but not in the source program being used, you can refer to the DECSYSTEM-20 (TOPS-20) Batch Reference Manual.

CONVENTIONS USED IN THIS MANUAL

The following is a list of symbols and conventions used in this manual.

dd-mmm-yy hh:mm	A set of alphanumeric characters that indicates date and time; e.g., 2-APRIL-78 15:30 or 2-APR-78 13:30. Time of day is represented by a 24-hour notation; 15:30 means 3:30 P.M.
filename.typ.gen	The name, type, and generation number of a file. The name can be 1 to 39 alphanumeric characters in length and the type can also be 1 to 39 alphanumeric characters in length. The name and type must be separated by a period. The generation number can be any positive integer, up to and including 2(17-1). The type and the generation numbers are optional, but if both are present, they must be separated by a period. Refer to the Glossary for the definitions of these terms.
hh:mm:ss	A set of numbers representing time in the form hours:minutes:seconds. Leading zeros can be omitted, but colons must be present between two numbers. For example, 5:35:20 means 5 hours, 35 minutes, and 20 seconds.
jobname	The name that is assigned to a job. It can contain up to six characters. Refer to the Glossary for the definition of a job.
n	A number that specifies either a required number or an amount of things such as cards or line-printer pages. This number can contain as many digits as are necessary to specify the amount required, e.g., 5, 13, 219, etc.
t	A number representing an amount of time, usually in minutes. This number can contain as many digits as are necessary to specify the amount of time required, e.g., 7, 40, 120, etc.
x	An alphanumeric character.

SYMBOLS USED IN THIS MANUAL

Symbol	Meaning
@TYPE	Anything you type on your terminal appears in red. Anything the system prints on your terminal appears in black.
RET	Press the key labeled RETURN or CR.
DEL	Press the key labeled DELETE or RUBOUT.
ESC	Press the key labeled ESC, ESCAPE, ALT, or PRE.
TAB	Press the key labeled TAB.
SP	Press the space bar once.
[]	Brackets enclose all optional arguments.
()	Parentheses enclose the name or value of an argument.

GLOSSARY

Term	Definition
ALGOL	ALGOrithmic Language. A scientifically oriented language that contains a complete syntax for describing computational algorithms.
Alphanumeric	Any of the letters of the alphabet (uppercase A through Z and lowercase a through z) and the numerals (0 through 9).
ASCII Code	American Standard Code for Information Interchange. Its 7-bit code is used to create a series of alphanumeric or special symbols.
Assemble	To prepare a machine-language program from a symbolic-language program by substituting absolute operation codes for symbolic operation codes and absolute or relocatable addresses for symbolic addresses.
Assembler	A program which accepts symbolic code and translates it into machine instructions, item by item. The assembler for TOPS-20 is called the MACRO assembler.
Assembly Language	The machine-oriented symbolic programming language. The assembly language for TOPS-20 is MACRO. MACRO statements are equivalent to one or more machine instructions.
Assembly Listing	A printed list which is the byproduct of an assembly run. It lists in logical-instruction sequence all details of a routine showing the coded and symbolic notation next to the actual assigned notations established by the assembly procedure.
BASIC	Beginner's All-purpose Symbolic Instruction Code. A timesharing computer programming language that is used for direct communication between terminal units and remotely located computer centers. BASIC employs English-like terms, is relatively easy to use, and has a wide range of applications.
Batch Processing	The technique whereby a computer executes one or more programs in your absence.

Term	Definition
Card	A punched card with 80 vertical columns representing 80 characters. Each column is divided into two sections, one with character positions labeled 0 through 9, and the other with positions 11 and 12. The 11 and 12 positions are not labeled and are also referred to as the X and Y zone punches, respectively.
Card Column	One of the vertical lines of punch positions on a punched card.
Card Field	A fixed number of consecutive card columns assigned to a unit of information.
Card Row	One of the horizontal lines of punch positions on a punched card.
Central Processing Unit (CPU)	The portion of the computer that contains the arithmetic, logical, control, and I/O interface circuits.
Character	One symbol of a set of elementary symbols such as those corresponding to the keys on a typewriter. The symbols usually include the decimal digits 0 through 9, the letters A, a through Z, z, punctuation marks, a space, operation symbols, and any other special symbols which a computer may read, store, or write.
COBOL	Common Business Oriented Language. A high-level source language widely used in business or commercial applications.
Command	An instruction that causes the computer to execute a specified operation.
Compile	To produce a machine- or intermediate-language routine from a routine written in a high-level language. A high-level language is user-oriented and one in which single statements may result in more than one machine-language instruction, e.g., FORTRAN, COBOL or ALGOL.
Compiler	A system program which translates a high-level source language into a language suitable for a particular machine. A compiler converts a source-language program into intermediate- or machine-language. Some compilers used on TOPS-20 are: ALGOL, COBOL, FORTRAN.
Computer	A device with self-contained memory capable of accepting information, processing the information, and outputting results.
Computer Operator	A person who has access to all software elements of a system and performs operational functions such as: loading a tape transport, placing cards in the card reader input hopper, removing printouts from the printer rack, etc.
Continuation Card	A punched card which contains information that was started on a previous punched card.

Term	Definition
Control File	The file made by you that directs Batch in the processing of your job.
CPU	See Central Processing Unit.
Cross-Reference Listing	A printed listing that identifies each reference of an assembled program with a specific label. This listing is provided immediately after a source program has been assembled.
Data	A general term used to denote any or all numbers, letters, and symbols, or facts that refer to or describe an object, idea, condition, situation, or other factors. It represents basic elements of information which can be processed or produced by a computer.
Debug	To locate and correct any mistakes in a computer program.
Disk	A form of a mass-storage device in which information is stored in named files.
Execute	To interpret an instruction or set of instructions and perform the indicated operation(s).
File	An ordered collection of 36-bit words composing computer instructions and/or data. A file can be of any length, limited only by the available space on the storage device and your maximum space allotment on that device.
Filename	A name of 1 to 39 alphanumeric characters chosen by the user to identify a file. Note that some commands only accept up to 6-character filenames.
File Type	A string of 1 to 39 alphanumeric characters, usually chosen to describe the class of information in a file. The file type must be separated from the filename by a period, e.g., FOR01.DAT. Note that some commands only accept up to 3-character file types.
FORTTRAN	FORMula TRANslator. A procedure-oriented programming language that was designed for solving scientific problems. The language is widely used in many areas of engineering, mathematics, physics, chemistry, biology, psychology, industry, the military, and business.
Generation Number	A number associated with a file within the file directory that is incremented each time the file is changed through editing, etc.
Job	The entire sequence of tasks performed between login and logout at an interactive terminal, with a card deck, or at an operator's console.

Term	Definition
Label	A symbolic name used to identify a statement in the control file or in a magnetic tape file or in a volume.
Log File	A file into which Batch writes a record of your entire job. This file may be printed as the final step in Batch's processing of a job.
MACRO	See Assembly Language.
Mounting a Device	A request to assign an I/O device via the operator.
Object Program	The program which is the output of compilation or assembly. Often the object program is a machine-language program ready for execution.
Password	The secret word assigned to you that, along with your user name, uniquely identifies you to the system.
Peripheral Device	Any unit of equipment, distinct from the central processing unit, which can provide the system with outside storage or communication.
Program	The complete sequence of machine instructions and routines necessary to resolve one or more computational tasks.
Programming	The science of translating a problem from its physical environment to a language that a computer can understand and obey; also, the process of planning the procedure for solving a problem. This may involve, among other things, the analysis of the problem, preparation of a flowchart, coding of the problem, establishing input-output formats, establishing testing and checkout procedures, allocation of storage, preparation of documentation, and supervision of the running of the program on a computer.
Queue	A list of jobs to be scheduled or run according to system-, operator-, or user-assigned priorities, for example, the Batch input queue is the list of jobs to be processed by Batch.
Software	The totality of programs and routines used by a computer. Examples include compilers, assemblers, operator programs, service routines, utility routines, and subroutines.
Source Deck	A card deck that constitutes a computer program in symbolic language.
Source Language	The original form in which a program is prepared prior to its processing by the computer to produce the object-language program.

Term	Definition
Source Program	A computer program written in a language designed for humans to use to express procedures or problem formulations. A translator (assembler, compiler, or interpreter) is used to perform the mechanics of translating the source program into an object- or machine-language program that can be run on a computer.
System	The collection of programs which schedules and controls the computing facility.
System Command	An instruction to the system to perform an operation. The system commands for the TOPS-20 are described in the DECSYSTEM-20 (TOPS-20) User's Guide.
System Program	A program generally available to users, administrators, or operators for performing some specific function. Examples are a FORTRAN compiler or a text editor.
Terminal	A device containing a keyboard, similar to a typewriter, and a printing or display mechanism employed to establish communications with a computer.
User Name	A name composed of from 1 to 39 alphanumeric characters (normally your surname) that identifies you and your logged-in directory.

CHAPTER 1

INTRODUCTION

1.1 WHAT BATCH IS

Many data processing jobs may require long running times and may make few demands of you. Ideally these jobs should be run in your absence when the computer is not busy with other tasks. This ideal is met by the TOPS-20 Batch system.

Batch is a group of programs that allows you to submit a job to the TOPS-20 system on a leave-it basis. (Refer to the DECSYSTEM-20 (TOPS-20) Batch Reference Manual for a complete description of programs that constitute the Batch system.) You may build and submit your job in one of two ways:

1. By entering your data directly to an interactive computer system by means of a timesharing terminal.
2. By entering your data from punched-cards to the interactive system. The cards are given to an operator who, at an appropriate time in his schedule, enters them into the computer through a card reader.

One advantage of Batch processing with an interactive system is that the interactive capabilities may be employed to greatly reduce the amount of time required to prepare the job for entry. By using a terminal to enter and edit the program items and data to be processed, you can bypass the tedious chore of preparing card decks. However, if desired, you can prepare traditional card decks by employing the punched card facilities of the system. In either case, the information to be entered is prepared as if it were to be processed as a normal job from an interactive terminal. The only added requirement is that special commands are entered with the job to direct the system in your absence. In other words, you anticipate the questions the system normally asks and you answer them when you enter the interactive job.

After preparing the job, you are free to leave the system. Upon accepting the job, the system classifies it in terms of size, running time, the need for peripherals, etc. This classification is used as the basis for determining when the job is to be run. Large jobs may, therefore, be set aside until smaller or more urgent jobs are finished.

Some of the jobs that are commonly processed through the TOPS-20 Batch system are those that:

1. are frequently run for production
2. are large and long running

INTRODUCTION

3. require large amounts of data
4. need no actions by you when the jobs are running

Batch allows you to submit your job to the computer through either an operator or a timesharing terminal, and to receive your output from the operator when the job has finished. Output is never returned to your timesharing terminal even if your job is entered from one. Instead, it is sent to a peripheral device (normally the line printer) at the computer site and returned to you in the manner designated by the installation manager.

1.2 HOW TO USE BATCH

To use the Batch system to process your job, you must create a control file. A control file consists of various commands that tell the TOPS-20 system what you want to process. The control file commands can be created as a disk file or as card input and can consist of:

1. System commands (see the DECSYSTEM-20 (TOPS-20) User's Guide),
2. System program commands to system programs, and
3. Batch commands (see Chapters 2 and 3).

These commands, when submitted to the operating system, must be in a particular order so that your Batch job will execute correctly.

The steps that you must take to create a control file from a timesharing terminal are described in Chapter 2. The steps to take to create a control file from cards are described in Chapter 3.

1.2.1 Running Your Job

After you submit the job, it waits in a queue with other jobs until Batch schedules it to run under guidelines established by the installation manager. Several factors affect how long your job waits in the queue, for example, its estimated execution time and the priority of your job compared with other waiting jobs.

When the job is started, Batch reads the control file to determine what actions are necessary to complete the job. For example, if there are commands to the system programs, Batch issues the commands to those programs. Any output produced as a result of those commands is stored in a log file for listing later. With adequate planning, the control file can also provide for corrective actions in the event of errors.

As each step in the control file is performed, Batch records it in a log file. For example, if a system command such as COMPILE is executed, Batch passes it to the system and writes it in the log file. The system response is also written in the log file. Batch writes in the log file any response from your job that would have been written on the terminal if the job were run interactively.

INTRODUCTION

1.2.2 Receiving Your Output

Your program output will be returned to you in the form that you specified by the commands in your control file. This is normally the line-printer listing, but may also be output on magnetic tape or disk. When your output is directed to the line printer, you may specify, in the SUBMIT command to the Batch process, the approximate number of pages (/PAGE: switch) that you require (to help Batch restrain runaway programs).

If your Batch job is submitted through a timesharing terminal, the log file is written and saved on disk in your directory and printed on the line printer. If your Batch job is submitted on card input, the log file is written on disk in your directory, printed on the line printer, and then deleted from your directory.

1.2.3 Recovering from Errors

If an error occurs in your job, either within a program that is executing or within the control file, Batch writes the error message in the log file and usually terminates the job. You can, however, include commands in the control file to direct Batch to branch to recovery sequences in the event of an error and thereby allow completion of the job. The effectiveness of error recovery is dependent on your ability to predict potential trouble spots within the program or within commands used in the control file. (Refer to the DECSYSTEM-20 (TOPS-20) Batch Reference Manual for detailed descriptions on error recovery for Batch jobs.)

1.3 SUMMARY

The steps that you must perform to enter a job to the computer through Batch are as follows:

1. Create a control file either on cards or from a terminal.
2. Submit the job to Batch, either indirectly via the operator (for a card job) or directly from a terminal.
3. Obtain and examine the log file listing and the job output to determine if the desired results were obtained.

Sample jobs run through Batch from cards and from a terminal are shown in Chapter 5.

CHAPTER 2

ENTERING A BATCH JOB FROM A TERMINAL

When you submit a job to Batch from a timesharing terminal, you must create a control file that Batch can use to run your job. The control file contains all the commands that you would use to run your job if you were running under timesharing. For example, if you wanted to compile and execute a program called MYPROG.CBL, the typeout on a timesharing terminal would appear as follows:

```

  ESC
  ↓
@COMPILE (FROM) MYPROG.CBL RET (Your request)
COBOL:MYPROG [MYPROG.CBL] (The system's reply)

EXIT

  ESC
  ↓
@EXECUTE (FROM) MYPROG.CBL RET (Your request)
LINK:LOADING
[LNKXCT MYPROG EXECUTION] (The system's reply)

EXIT
@
```

To create a control file to tell Batch to run the same, you would create the following:

```

  ESC
  ↓
@CREATE (FILE) MYFILE.CTL RET
INPUT: MYFILE.CTL.1
00100 @COMPILE MYPROG.CBL RET
00200 @EXECUTE MYPROG.CBL RET
00300 $
      ESC
      ↓
    *E RET

[MYFILE.CTL.1]

@
```

When the job is run, the commands are passed to the system to be executed. The commands and their replies from the system are written in the log file so that the entire dialogue shown in the first example above appears in the log file.

ENTERING A BATCH JOB FROM A TERMINAL

2.1 CREATING THE CONTROL FILE

To create a control file and submit it to Batch from a terminal, you must perform the following steps:

1. LOGIN to the system as a timesharing user
2. Create a control file on disk using EDIT
3. Submit the job to Batch using the system command SUBMIT

You can then wait for your output to be returned at the designated place.

After you have logged into the system as you normally would to start a timesharing job, you can use EDIT so that you can create your control file.

The control file can contain TOPS-20 EXEC commands, program commands, data that would normally be entered from a terminal, and Batch commands. The Batch commands are described in Section 2.3. What you write in the control file depends on what you wish your job to accomplish. An example of a job that you can enter for Batch processing is as follows:

1. Compile a program that is on disk.
2. Load and execute the program using data from a file already on disk.
3. Print the output on the line printer.
4. Write the output into a disk file also.
5. Compile a second program.
6. Load and execute the second program using the data output from the first program.
7. Print the output from the second program.

The control file that you would create for the preceding job would appear as follows:

```

      ESC
      ↓
@CREATE (FILE) MYFILE.CTL RET
INPUT: MYFILE.CTL.1
00100 @COMPILE MYPROG.FOR/COMPILE RET
00200 @EXECUTE MYPROG.FOR RET
00300 @COMPILE PROG2.FOR/COMPILE RET
00400 @EXECUTE PROG2.FOR RET
00500 $
      ↓
      ESC
      ↓
*E RET
[MYFILE.CTL.1]
@
```

Include statements in your programs (rather than in the control file) to read the data from the disk files and write the output to the printer and the disk. The output to the line printer is written along with your log file as part of the total output of your job.

ENTERING A BATCH JOB FROM A TERMINAL

If an error occurs in your job, Batch will not continue but will terminate the job. To avoid having your job terminated because an error occurs, you can specify error recovery in the control file using special Batch commands. Error recovery is described in Section 2.4.

Any system command that you can use in a timesharing job can be used in a Batch job with the following exceptions. The ATTACH and SET TIME-LIMIT commands are illegal in a Batch job. If you include either of these commands in your job, Batch will process the command and the TOPS-20 command processor (the EXEC) will place an error message into your log file. Your Batch job will terminate unless you specify error recovery.

Do not include a LOGIN command in your control file since Batch logs the job for you. If you put in a LOGIN command, your job will be terminated. In addition, you do not need to include a LOGOUT command. Batch will log out your job automatically when it reaches the end of your control file.

2.1.1 Format of Lines in the Control File

Since you can put TOPS-20 EXEC commands, program commands, and Batch commands, as well as data, into the control file, you have to tell Batch what kind of line it is reading. Batch determines the line it is reading as a command, data, or comment by the first nontab or nonblank character. The first character in each line should be one of the characters described below.

To put a system command or Batch command into your control file, you must put an at sign (@) in the first column and follow it immediately with the command.

To put a command string of a system program or user program into your control file, put an asterisk (*) in column 1 and follow it immediately with the command string. For the format of command strings, refer to the manual for the specific program that you wish to use.

If you want to include in the control file a command to a system program that does not accept carriage return as the end of the line, e.g., DDT, you must substitute an equal sign (=) for the asterisk so that Batch will suppress the carriage return at the end of the line.

To include in the control file data for your program, write it as you would data that is read from a separate file. If your program prompts you with an asterisk (during timesharing) for data input, then you should include the asterisk as the first character before the data in your control file. If your program does not require an asterisk (*) prompt, you do not have to include it with your data.

Comments can also be included in the control file either as separate lines or on lines containing other information. To include a comment on a separate line, you must put an exclamation point (!) in column 1 and follow it with the comment. To add a comment to a line after your data, you must precede the comment with an exclamation point (!).

If you put in the first column of the line any special characters other than those described, you may get unexpected results because Batch interprets other special characters in special ways.

ENTERING A BATCH JOB FROM A TERMINAL

The following example illustrates a control file, using some of the characters described above, and the resulting log file. The example uses the TOPS-20 FILCOM utility to compare two files.

```
{This batch job generates a FILCOM of two files,
!Run FILCOM and then give it a command,
@FILCOM
*TTY:=FILE,QXT,FILE,TXT
**C

13:11:02 BAJOB  BATCON version 103(3000) running EXAM2 sequence 6152 in stream 1
13:11:02 BAFIL  Input from PS:<SMITH>EXAM2,CTL,1
13:11:02 BAFIL  Output to PS:<SMITH>EXAM2,LOG
13:11:02 BASUM  Job parameters
                  Time:00:05:00  Unique:YES  Restart:NO  Output:NOLOG

13:11:02 MONTR
13:11:02 MONTR  SYSTEM 2102 DEVELOPMENT SYSTEM, TOPS-20 Monitor 4(1707)
13:11:02 MONTR  @LOGIN SMITH 341
13:11:06 MONTR  Job 40 on TTY225 24-May-78 13:11:06
13:11:07 MONTR  @
13:11:07 MONTR  {CONNECTED TO PS<SMITH>}
13:11:07 MONTR  @SET TIME=LIMIT 300
13:11:09 MONTR  @
                  {This batch job generates a FILCOM of two files,
                  !Run FILCOM and then give it a command,
                  @FILCOM
13:11:09 MONTR
13:11:14 USER  **TTY:=FILE,QXT,FILE,TXT
13:11:14 USER  File 1) DSK:FILE,QXT      created: 1307 24-MAY-1978
13:11:14 USER  File 2) DSK:FILE,TXT    created: 1308 24-MAY-1978
13:11:14 USER
13:11:14 USER  1)1 This file contains a spelling error,
13:11:15 USER  ***
13:11:15 USER  2)1 This file does not contain a spelling error,
13:11:15 USER  *****
13:11:15 USER
13:11:15 USER  %files are different
13:11:15 USER
13:11:16 USER  **C
13:11:17 MONTR  *C
13:11:18 MONTR  @
13:11:18 MONTR  @*C
13:11:18 MONTR  @LOGOUT
13:11:20 MONTR  Killed Job 40, User SMITH, Account 341, TTY 225,
13:11:20 MONTR  at 24-May-78 13:11:19, Used 010:0 in 010:13
```


2.2 SUBMITTING THE JOB TO BATCH

After you have created the control file and saved it on disk, you must enter it into the Batch queue so that it can be run. All programs and data that are to be processed when the job is run must be made up in advance or be generated during the running of the job. You can have them on magnetic tape, but if you do, you must include the TOPS-20 commands TMOUNT, UNLOAD, and DEASSIGN in your control file so that the operator will mount and dismount the tape(s) to be read. (Refer to Chapter 5, for examples of a control file with these three commands.)

If your programs and data reside on an on-line disk, you need not include the TMOUNT, UNLOAD or DEASSIGN command as there is no action required by the operator.

ENTERING A BATCH JOB FROM A TERMINAL

You enter your job in Batch's queue by means of the TOPS-20 SUBMIT command. This command has the form:


control-file-specification is the name you have given to the control file you created. you must specify the filename of the control file. You can specify a file type or, if you do not, the EXEC (TOPS-20 command processor) will assume a file type of .CTL.

/switches are switches to Batch to tell it how to process your job and what your output will look like. Some of the switches that can be used with the SUBMIT command are described in the following section. (Refer to the DECSYSTEM-20 (TOPS-20) Batch Reference Manual for a complete description of all available switches.)

2.2.1 Switches

You use the switches to define limits for your job. Such limits as pages of output and the time that your job will run can be specified as switches. Each switch can be specified only once in a SUBMIT command. You can put a switch anywhere in the command string.

/AFTER:hh:mm Switch

If you do not want Batch to run your job until after a certain time or until after a certain number of minutes have elapsed since the job was entered, you can include the /AFTER switch in the SUBMIT command string. To run the job after a specified time of the day, you must specify the time in the form hh:mm (for example, /AFTER:12:00 to run a job after noon). To run the job after a given amount of time has elapsed, specify the time in the form +hh:mm (for example, /AFTER:+1:00 to run the job an hour from now). If you omit the switch, Batch will schedule your job as it normally would using its defaults. If you omit the colon and/or value, the EXEC will respond with an error message and terminate the SUBMIT command.

/PAGE:n Switch

Normally, Batch allows your job to print up to 200 pages. Included in this number are the log file and any compilation listings that you may request. If you need more than 200 pages for your job, you must include the /PAGE switch in the SUBMIT command to indicate the approximate number of pages that your job will print. If you include the switch without the colon and a value, The EXEC will assume that you will print up to 2000 pages. If your output exceeds the number that you specified in the /PAGE switch, the excess output will be lost and the message ?LPTPLE PAGE LIMIT EXCEEDED will be printed.

ENTERING A BATCH JOB FROM A TERMINAL

However, even if you exceed the maximum, the first 10 pages of the log file will be printed.

/TIME:hh:mm:ss Switch

Normally, Batch allows your job to use up to five minutes of central processor time. Central processor (CPU) time is the amount of time that your job runs, not the amount of time that it takes Batch to process your job. If you need more than five minutes of CPU time, you must include the /TIME switch in the SUBMIT command to indicate the approximate amount of time that you will need. If you specify the switch without the colon and a value, Batch will assume that you need one hour of CPU time. If you do not specify enough time, Batch will terminate your job when the time is up.

The value in the /TIME switch is given in the form hh:mm:ss (hours:minutes:seconds). If you specify only one number, Batch assumes that you mean minutes. Two numbers separated by a colon are assumed to mean hours and minutes. All three numbers, separated by colons, mean hours, minutes, and seconds. For example:

/TIME:25	means 25 minutes
/TIME:1:25	means 1 hour and 25 minutes
/TIME:1:25:00	means 1 hour and 25 minutes and no seconds

2.2.2 Examples of Submitting Jobs

The following are sample jobs entered to Batch by means of the SUBMIT command.

Example 1:

This control file consists of commands to compile FORTRAN program, print a listing, and execute it.

```
      ESC
      ↓
@CREATE <FILE> MYFILE.CTL RET
INPUT: MYFILE.CTL.1
00100 @COMPILE MYPROG.FOR/LIST/COMPILE RET
00200 @EXECUTE MYPROG.FOR RET
00300 $
      ↑
      ESC
      ↓
*E RET

[MYFILE.CTL.1]

@
```

After the control file to compile and execute the FORTRAN program has been created and saved, you must submit the job to Batch.

```
      SP
      ↓
@SUBMIT MYFILE.CTL RET
```

ENTERING A BATCH JOB FROM A TERMINAL

When the EXEC reads this SUBMIT command, it assumes the following:

1. The control filename and type are MYFILE.CTL.
2. The name of the job is MYFILE.
3. The log file will be named MYFILE.LOG.
4. Both the control file and the log file will be saved in your disk area.
5. 200 is the maximum number of pages to be printed (/PAGE:200).
6. The maximum amount of CPU time is 5 minutes (/TIME:5:00).

Example 2:

The next example shows the control file that was created at the beginning of this chapter being submitted to Batch.

```
      SP  
      ↓  
@TYPE MYFILE.CTL RET  
00100 @COMPILE MYPROG.FOR/COMPILE  
00200 @EXECUTE MYPROG.FOR  
00300 @COMPILE PROG2.FOR/COMPILE  
00400 @EXECUTE PROG2.FOR  
@
```

After you have saved the control file, you can submit the job to Batch.

```
      SP  
      ↓  
@SUBMIT MYFILE/JOBNAME:MYJOB/TIME:20/PAGE:750/AFTER:10:00 RET
```

When the EXEC reads this request, it assumes the following:

1. The name of the job is MYJOB.
2. The name of the control file is MYFILE.CTL.
3. The log file will be named MYFILE.LOG.
4. The log file will be left in your disk area after it is printed.
5. The control file will be left in your disk area.
6. 750 is the maximum number of pages that can be printed (/PAGE:750).
7. The maximum amount of CPU time that the job can use is 20 minutes (/TIME:20:00).
8. The job will process only after 10:00 in the A.M.

If you made an error in the SUBMIT command when you submitted either of these jobs, the EXEC will type an error message on your terminal to explain your error so that you can correct it.

ENTERING A BATCH JOB FROM A TERMINAL

2.3 BATCH COMMANDS

You can write certain Batch commands in the control file to tell Batch how to process your control file. Each of these commands must be preceded by an at sign (@) so that Batch will recognize it. The most commonly used Batch commands are described in the following sections, but not all Batch commands are described here. For a description of all commands, refer to the DECSYSTEM-20 (TOPS-20) Batch Reference Manual.

2.3.1 The @IF Command

You can include the @IF command in your control file to specify an error-recovery procedure to Batch or to specify normal processing if an error does not occur. The @IF statement has the forms:

@IF (ERROR) statement (The parentheses must be included.)

@IF (NOERROR) statement (The parentheses must be included.)

where

statement is a command to the system, to a program, or to Batch.

An example of the @IF (ERROR) command follows:

```
!DO A DIRECTORY IF AN ERROR OCCURS
@IF (ERROR) @VDIRECTORY
.
.
.
```

An example of the @IF (NOERROR) command follows:

```
!IF NO ERROR OCCURS, GIVE A SECOND LINE OF INPUT
@IF (NO ERROR) *FILE.SCM=A.TXT,B.TXT
```

The @IF command can be used in two ways as shown in its two forms. You can include the @IF (ERROR) command in your control file at the place where you suspect an error may occur. The @IF (ERROR) command must be the next command in your control file (that is, the next line which begins with an at sign (@)) after an error occurs; otherwise, Batch will terminate your job. In the @IF (ERROR) command, you direct Batch to either go back or forward in your control file to find a line that will perform some task for you or that will direct the system or any other program to perform some task for you.

You can use the @IF (NOERROR) command to direct Batch or the system to perform tasks for you when an error does not occur at the point in your control file where you place the @IF (NOERROR) command. Thus, if you expect that an error will occur in your program, you can include an @IF (NOERROR) command to direct Batch in case the error does not occur, and then put the error-processing lines immediately following the command. Refer to Section 2.4 for more examples of using @IF (NOERROR) and @IF (ERROR).

If an error occurs and Batch does not find an @IF command as the next command line in the control file, Batch terminates the job.

ENTERING A BATCH JOB FROM A TERMINAL

2.3.2 The @ERROR Command

With the @ERROR command, you can specify to Batch the character that you wish to be recognized as the beginning of an error message. Normally, when Batch reads a message that begins with a question mark (?), it assumes a fatal error has occurred and terminates the job, unless you have specified error recovery (refer to Section 2.4). If you wish Batch to recognize another character (in addition to the question mark) as the beginning of a fatal error message, you must specify the character in the @ERROR command. The character specified may not be a control character, an exclamation point (!) or a semicolon(;). The exclamation point will be interpreted as the comment character and will not function as the error signal character. This command has the form:

@ERROR character

where

character is a single ASCII character

If you do not specify a character in the @ERROR command, Batch uses only the standard error character, the question mark. When a line that begins with the character you specify in the @ERROR command is output to the Batch job by the system, a system program, or is issued by Batch itself, Batch treats the line as a fatal error and terminates the job, exactly as it would if the line were preceded by a question mark. Any messages preceded by other characters will not be recognized by Batch as errors.

If you do not include the @ERROR command in your control file, Batch will recognize only the question mark as the beginning character of a fatal error message.

An example of the @ERROR command follows.

```
.  
.  
.  
@ERROR %  
.  
.  
.  
@ERROR
```

In this example, you specify in the middle of the control file that you want Batch to recognize the question mark (?) and the percent sign (%) as the beginning character of a fatal error from that point in the control file. Further on in the control file, you tell Batch to go back to recognizing only the question mark as the beginning of a fatal error message.

2.3.3 The @NOERROR Command

You can use the @NOERROR command to tell Batch to ignore all error messages issued by the system, system programs, and Batch itself. The @NOERROR command has the form:

@NOERROR

When Batch reads the @NOERROR command, it ignores any error messages that would normally cause it to terminate your job. The only

ENTERING A BATCH JOB FROM A TERMINAL

exception is the message ?TIME LIMIT EXCEEDED. Batch will always recognize this as an error message, give you an extra 10% of your allotted time, and terminate your job.

You can use @NOERROR commands in conjunction with @ERROR commands in the control file to control error reporting. For example, if you wish to ignore errors at the beginning and end but not in the middle of the control file, place @ERROR and @NOERROR commands at the appropriate places in the control file. In addition, you can also specify which messages must be treated as fatal errors.

```
.  
. .  
@NOERROR  
. .  
@ERROR %  
. .  
@ERROR  
. .  
@NOERROR
```

The first command tells Batch to ignore all errors in your job. The second command tells Batch to recognize as errors any message that starts with a question mark (?) and a percent sign (%). You change the error reporting with the next command to tell Batch to go back to recognizing only messages that begin with a question mark as fatal. The second @NOERROR command tells Batch to ignore all error messages again. If the ?TIME LIMIT EXCEEDED message is issued at any time, Batch will print the message, extend the time by 10%, and then terminate the job.

2.3.4 The @GOTO Command

You can include the @GOTO command in your control file to direct Batch to skip over lines in the control file to find a specific line. The @GOTO command has the form:

```
@GOTO label
```

where

label is a one- to six-character alphanumeric label for a statement. It must be followed by two colons (::).

An example of the @GOTO command follows.

```
.  
. .  
@GOTO ABC  
. .  
ABC::@DIRECTORY
```

ENTERING A BATCH JOB FROM A TERMINAL

You can use the @GOTO command as the statement in an @IF command (refer to Section 2.3.1) to aid you in error processing. For example:

```
.  
. .  
@IF (ERROR) @GOTO ABC  
. .  
ABC::@TYPE MYPROG
```

When Batch encounters a @GOTO command in the control file, it searches forward in the control file to find the label specified in the @GOTO command. Batch then resumes processing of the control file at the line which has the specified label. If the label is not found, Batch will issue the message

```
? BTNCNF COULD NOT FIND LABEL xxxxxx
```

and the job will be terminated.

If you do not include a @GOTO command in the control file, Batch reads the control file sequentially from the first statement to the last.

2.3.5 The @BACKTO Command

You can use the @BACKTO command to direct Batch to search back in the control file for a line with a specified label. The @BACKTO command has the form:

```
@BACKTO label
```

where

label is a one- to six-character alphanumeric label for a statement. It must be followed by two colons (::).

An example of the @BACKTO command follows.

```
.  
. .  
ABC::@DIRECTORY  
. .  
@BACKTO ABC
```

Normally, Batch reads the control file line by line and passes the commands and data to the system and your program. When you put a @BACKTO command into the control file, you tell Batch to interrupt the normal reading sequence and to search back in the control file to find a line containing the label specified in the @BACKTO command. The @BACKTO command searches for the label you specified, starting from the beginning of the file and ending at the place the command was given. When the labeled line is reached, Batch executes the line and continues from that point.

If Batch cannot find the labeled line, it terminates your job.

ENTERING A BATCH JOB FROM A TERMINAL

2.4 SPECIFYING ERROR RECOVERY IN THE CONTROL FILE

If you do not specify error recovery when an error occurs in your job, Batch terminates the job. You can specify error recovery in the control file by means of the Batch commands, especially the @IF Batch command. You must put the @IF command at the point between programs in the control file where an error may occur. When an error occurs, Batch skips over all lines in the control file until it encounters a line beginning with an at sign (@). If this line contains an @IF command, the @IF command is processed and the job continues. If this line does not contain an @IF command, the job is terminated. Therefore, if a Batch job is to recover from an error successfully, the @IF command must be placed in the control file where the error is expected to occur but before any other commands preceded by the @ sign. Thus, if you have a program that you are not sure is error free, you can include an @IF command to tell Batch what to do if an error occurs, as shown in the following example.

```
@COMPILE MYPROG.FOR
@IF (ERROR) statement
```

```
.
```

In either the @IF (ERROR) or the @IF (NOERROR) command, you should include a statement that tells Batch what to do. You can use any monitor command or Batch command. If you wish to simply ignore the error without taking any special action, you may use a comment as the statement. The @GOTO and @BACKTO commands are also commonly used for this purpose. Refer to Sections 2.3.4 and 2.3.5 for descriptions of these commands. Be sure, if you use @GOTO or @BACKTO in the @IF command, that you supply a line in the control file that has the label that you specified in the @GOTO or @BACKTO command.

Two sample jobs are shown below. The first shows the @IF (ERROR) command and the @GOTO command to specify error recovery. The second example shows the use of the @IF (NOERROR) and @GOTO commands.

If you have a program that you are not sure will compile without errors, you can include another version of the same program in your job (that hopefully will compile) and tell Batch to compile the second program if the first has an error. You write the control file as follows.

```

      ESC
↓
@CREATE (FILE) MYFILE.CTL RET
INPUT: MYFILE.CTL.1
00100 @COMPILE /COMPILE MYPROG.FOR/LIST RET
00200 @IF (ERROR) @GOTO A RET
00300 @EXECUTE MYPROG.FOR RET
00400 @GOTO B RET
00500 A:!!CONTINUE RET
00600 @COMPILE /COMPILE PROG2.FOR/LIST RET
00700 @EXECUTE PROG2.FOR RET
00800 B:!!CONTINUE RET
00900
      ESC
↓
*E RET

[MYFILE.CTL.1]
@
```

ENTERING A BATCH JOB FROM A TERMINAL

When the job is run, Batch reads the control file and passes commands to the system. If an error occurs in the compilation of the first program, Batch finds the @IF (ERROR) command and executes the @GOTO command contained in it. The @GOTO command tells Batch to look for the line labeled A::. Thus, Batch skips lines in the control file until it finds label A and then passes commands to the batch job from that point. If an error does not occur while compiling MYPROG, the @GOTO A statement is not executed. Instead, MYPROG is executed and then Batch skips to the line labeled B::.

A variation of the above procedure is shown below using the @IF (NOERROR) command and the @GOTO command. The difference is that Batch skips the @IF (NOERROR) command if an error does occur, and performs it if an error does not occur. The following is the control file that you would create.

```

      ESC
↓
@CREATE (FILE) MYFILE.CTL RET
INPUT: MYFILE.CTL.1
00100  @COMPILE /COMPILE MYPROG.FOR/LIST RET
00200  @IF (NOERROR) @GOTO A RET
00300  @COMPILE /COMPILE PROG2.FOR/LIST RET
00400  @EXECUTE PROG2.FOR RET
00500  @GOTO B RET
00600  A::!CONTINUE RET
00700  @EXECUTE MYPROG.FOR RET
00800  B::!CONTINUE RET
00900
      ESC
↓
*E RET

[MYFILE.CTL.1]

@
```

When the job is run, Batch passes the COMPILE command to the system to compile the first program. If an error does not occur, the @IF (NOERROR) command and the @GOTO command are executed, Batch skips to the line labeled A, which is a comment, and passes commands to the Batch job from that point. The program MYPROG.FOR is executed and the end of the job is reached. If an error occurs while compiling MYPROG, Batch skips the @IF (NOERROR) command and continues reading the control file. PROG2.FOR is compiled and then executed. Batch is then told to go to the line labeled B, which is a comment line. The end of the job follows.

The examples shown above illustrate only two ways that you can use the @IF commands to specify error recovery in the control file. You can use any of the Batch commands or any system command that you choose to recover from errors in your job.

However, you do not have to attempt to recover from errors while your job is running. You can correct your errors according to the error messages in the log file when your job is returned to you, and then run your job again. The log file is described in Chapter 4.

CHAPTER 3

ENTERING A BATCH JOB FROM CARDS

When you enter a job to Batch from card input, you must create a control file on cards that is somewhat similar to a control file created on a timesharing terminal, but that contains some additional Batch commands. The card control file must tell Batch to start your job, the tasks or steps your job must take, and when to stop your Batch job. The tasks or steps in your Batch job can consist of calls to a system program, can identify and protect the control file, and can establish error recovery.

Your control card input to Batch may contain any combination of commands. These commands are in four groups as follows:

1. TOPS-20 system commands, which consist of commands in a format similar to what you would issue for the same command on a timesharing terminal. Examples of these commands are @COPY, @DEASSIGN, @PRINT, and @RENAME.
2. System program commands, which consist of commands that pertain to a system or user program. An example is the command to the FILCOM program to specify files to be compared.
3. Batch commands, as described in Chapter 2, Section 2.3.
4. Batch control card commands, some of which are listed below.

\$JOB	(See Section 3.2.1)
\$PASSWORD	(See Section 3.2.2)
\$EOJ	(See Section 3.2.3)
\$CREATE	(See Section 3.2.4)
\$-language	(See Section 3.2.5)
\$EXECUTE	(See Section 3.2.6)
\$DATA	(See Section 3.2.7)
\$EOD	(See Section 3.2.8)
\$TOPS20	(See Section 3.2.9)
\$ERROR	(See Section 3.2.10)
\$NOERROR	(See Section 3.2.10)

Not all of the available Batch control commands are listed above and described in Section 3.2. Refer to the DECSYSTEM-20 (TOPS-20) Batch Reference Manual for a complete description of all available Batch control card commands.

ENTERING A BATCH JOB FROM CARDS

3.1 BATCH CONTROL CARD FORMAT CONVENTIONS

The Batch control cards must contain a dollar sign (\$) in column 1 and a command that starts in column 2. The command must be followed by at least one space, which can then be followed by other information. (Refer to the individual description of each card for more information about it.)

A card with a TOPS-20 system command must contain an at sign (@) in column 1 followed immediately by the command. Any information that follows the command is in the format shown for the command in the DECSYSTEM-20 (TOPS-20) User's Guide. The \$TOPS20 Batch control card command must precede TOPS-20 system commands in your card deck. (Refer to Section 3.2.9 for the format and description of the \$TOPS20 card.)

A card with a command to a system program must contain an asterisk (*) in column 1 followed immediately by the command string.

Batch commands are formatted in the same manner as system commands; that is, an at sign (@) is punched in column 1 and the command immediately follows it. You must also place a \$TOPS20 card before Batch commands in the card deck to enable execution of these commands.

If you put any special characters other than those described above in the first column of a card, you may get unexpected results because Batch interprets other special characters in special ways.

If you have more information than will fit on one card, insert a hyphen (-) as the last nonspace character on the first card and continue the information on the second card.

Comments can also be included either on separate cards or on cards containing other information. If the entire card is to contain a comment, the card should contain a dollar sign (\$) in column 1 and an exclamation point (!) in column 2. The exclamation point (!) is called the comment character. If the card contains a command followed by a comment, only the exclamation point (!) should precede the comment. If the comment is too long to be contained on a single card, begin the next card with a dollar sign (\$) in column 1 and the exclamation point (!) in column 2 and then continue the comment.

3.2 BATCH CONTROL CARD COMMANDS

Eleven Batch control card commands are described in the following sections. Additional Batch control card commands are available and can be referred to in the Batch Reference Manual.

3.2.1 Starting a Job - The \$JOB Card

The \$JOB card is the first card in your card deck. The \$JOB card tells Batch whose job it is processing and, optionally, the name of the job, and any constraints that you want to place on the job. When Batch reads the \$JOB card, it begins the log file for your job.

ENTERING A BATCH JOB FROM CARDS

The \$JOB card has the form:

The diagram shows a rectangular box representing the \$JOB card. Inside this box, there is a smaller rectangle at the top left with a slanted top edge. This smaller rectangle is labeled "\$JOB user-name/switches". The rest of the box is a large empty area for the job description.

user-name is the name assigned to you by the installation to allow you to gain access to the DECSYSTEM-20. Normally, your user-name is your surname.

/switches are optional switches to Batch to tell it the constraints that you have placed on your job. They are described below.

/AFTER:dd-mmm-yy hh:mm Switch

If you do not want Batch to run your job until after a certain time and/or a certain day, you can include the /AFTER switch on your \$JOB card. The date and time are specified in the form dd-mmm-yy hh:mm (e.g., 16-APR-78 17:15). If you omit this switch, Batch schedules your job as it normally does; that is, Batch schedules your job based on the time required and other parameters.

/AFTER:+hh:mm Switch

If you do not want Batch to run your job until after a certain amount of time has elapsed since the job was entered, include this form of the /AFTER switch on the \$JOB card. The amount of time that the job must wait after it has been entered is specified in the form +hh:mm (e.g., +1:30). If this switch is not included, Batch will schedule the job as it normally does.

/JOBNAME:xxx Switch

You can set the name of the job by inserting this switch on your \$JOB card. The name can be 1 to 6 alphanumeric characters in length. If you omit this switch, Batch will create a unique name for your job. The name created by Batch is assigned to both your control file and your log file. Batch adds the file type .CTL to the control file and the type .LOG to the log file.

/PAGES:n Switch

Normally, Batch allows your job to print up to 200 pages. Included in this number are the log file and any compilation listings that you may request. If you need more than 200 pages for your job, you must

ENTERING A BATCH JOB FROM CARDS

include the /PAGES switch on the \$JOB card to indicate the approximate number of pages that your job will print. If your output exceeds either the maximum that Batch allows or the number that you specified in the /PAGES switch, the excess output will not be printed and the message ?LPTPLE PAGE LIMIT EXCEEDED will be written in the log file. However, even if you exceed the maximum, the first 10 pages of the log file will be printed.

NOTE

Do not arbitrarily enter a large PAGES value as this may delay execution of your Batch job.

/TIME:hh:mm:ss Switch

Normally, Batch allows your job to use up to five minutes of central processor (CPU) time. CPU time is the amount of time that your job runs in memory, not the amount of time that it takes Batch to process your job. If you need more than five minutes of CPU time, you must include the /TIME switch on the \$JOB card to indicate the approximate amount of time that you will need. If you do not specify enough time, Batch will terminate your job when the time is up. However, if you specify a large amount of time, Batch may hold your job in the queue until it can schedule a large amount of time for it.

The value in the /TIME switch is given in the form hh:mm:ss (hours:minutes:seconds). If you specify only one number, Batch assumes that you mean minutes. Two numbers separated by a colon (:) are assumed to mean hours and minutes. All three numbers, separated by colons mean hours, minutes, and seconds. For example:

/TIME:25 means 25 minutes
/TIME:1:25 means 1 hour and 25 minutes
/TIME:1:25:00 means 1 hour, 25 minutes, and no seconds

The following rules apply to all switches in the above list that require a time and/or date to be specified:

When you specify the time of day (hh:mm:ss)

1. You must not omit the colon (:) or colons.

When you specify a date (dd-mmm-yy)

1. You must not omit the hyphens.
2. You must specify both the day and the month as a minimum requirement.
3. You can abbreviate the month to a minimum of three letters, e.g., JUL for July.
4. If you omit the year, the current year will be used.

ENTERING A BATCH JOB FROM CARDS

5. If you omit the time from a date specification, the time is assumed to be midnight on the specified date. In the example below a current date and time of 20 April, 1978, 10AM will be assumed.

/AFTER:18:00	means 6 P.M. on April 20, 1978
/AFTER:3-May	means midnight on May 3, 1978
/AFTER:19-Apr 20:00	means 8 P.M on April 19, 1978

3.2.2 Identifying Yourself - The \$PASSWORD Card

You put the password that has been assigned to you on the \$PASSWORD card to tell Batch that you are an authorized user of the system.

In conjunction with the \$JOB card, the \$PASSWORD card identifies you to Batch and tells Batch to process your job. If you put a password on the \$PASSWORD card that does not match the password stored in the system for you, Batch will terminate your job. The \$PASSWORD card must be present and must immediately follow the \$JOB card.

The \$PASSWORD card has the form:

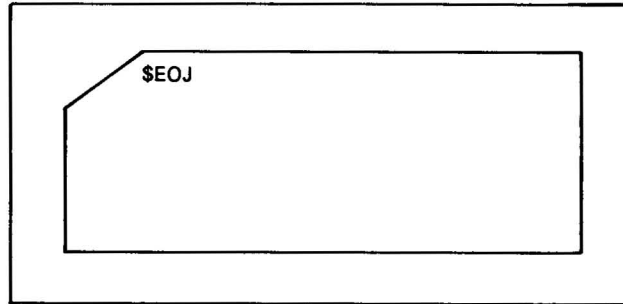
The diagram shows a rectangular card with a tab on the top-left corner. The text "\$PASSWORD password" is printed on the card.

password is a 1- to 39-character password that is stored in the system to identify you. There must be exactly one space between the end of the card name (\$PASSWORD) and the first character of your password.

ENTERING A BATCH JOB FROM CARDS

3.2.3 Ending a Job - The \$EOJ Card

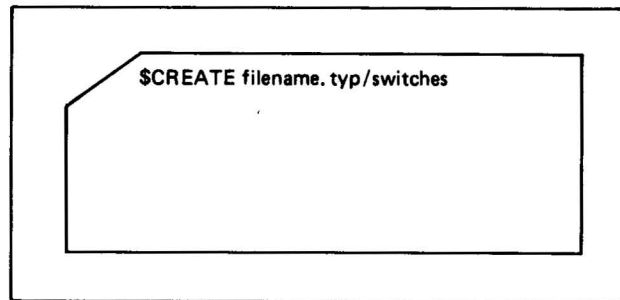
You must put the \$EOJ card at the end of the deck containing your complete job to tell Batch that it has reached the end of your job. If you omit the \$EOJ card, an error message will be issued. However, your job will still be scheduled and may be processed if another job follows it. The form of the \$EOJ card is shown below.



3.2.4 Creating a File - The \$CREATE Card

You can put the \$CREATE card in front of any program, data, or other set of information to make Batch copy the program, data, or information into a disk file. If the appropriate switch is included, Batch will also print this file on the line printer.

The form of the \$CREATE card is:



filename.typ specifies the optional filename and type you want Batch to put on the file it creates for your program or data. If you omit the filename and type, Batch will create a unique name for your file of the form CRxxxx, where xxxx represents a unique name generated by Batch.

/switches are switches to Batch to tell it how to read the cards in your deck. The switches are described below.

ENTERING A BATCH JOB FROM CARDS

/WIDTH:n Switch

Normally, Batch reads 80 columns on every card in your deck. You can make Batch stop reading at a specific column by means of the /WIDTH switch, where you indicate the number of column at which to stop. Thus, if you have no information in the last 10 columns of each card in your deck, you can tell Batch to read only up to column 70 by specifying

/WIDTH:70

/SUPPRESS Switch

When Batch reads the cards in your deck, it normally copies everything on the card up to column 80 (or up to any column you may specify on the /WIDTH switch). However, if you do not want trailing spaces copied (to save space on the disk, for example), you can tell Batch, by means of the /SUPPRESS switch, not to copy any trailing spaces into the disk file.

/PRINT Switch

The file currently being created on disk by Batch is listed on the line printer.

Examples

The simplest form of the \$CREATE card is:

\$CREATE

This card causes Batch to copy your deck into a disk file and to assign a unique name to it. All 80 columns of the cards are read and trailing spaces are copied into the file. The file is not printed.

The following is an example of a \$CREATE card.

\$CREATE MYFILE.CDS/WIDTH:50/PRINT

The deck that follows this card is copied into a disk file named MYFILE.CDS. When Batch reads the cards in the deck, it copies trailing spaces into the file, reading up to 50 columns. The disk file created from your cards will be printed on the line printer.

3.2.5 Compiling a Program - The \$-language Card

The \$-language card specifies the source program language of your program on cards. It is placed in front of your program. The \$-language card may be any of the following:

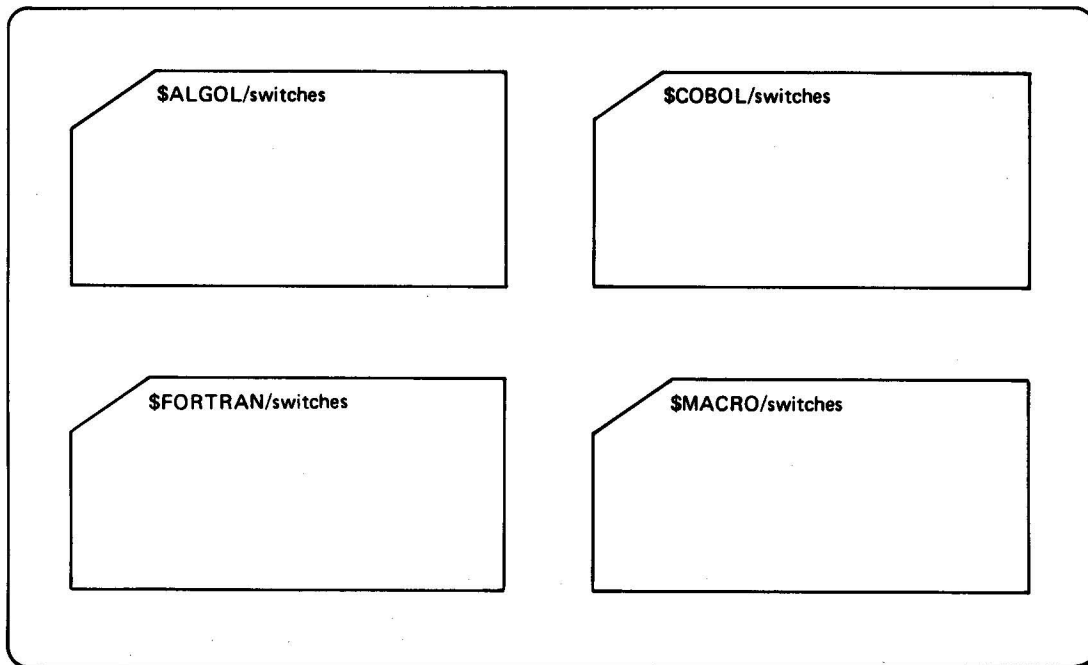
1. \$COBOL
2. \$FORTRAN
3. \$MACRO
4. \$ALGOL

ENTERING A BATCH JOB FROM CARDS

The \$-language card causes Batch to copy your source program into a disk file and compile it. You may then execute your program by using the \$EXECUTE card (Section 3.2.6) or the \$DATA card (Section 3.2.7). Optional information (/switches) may be included on the \$-language card to tell Batch how to read and compile your program.

When Batch copies your source program file onto disk, Batch assigns a unique filename for your program in the form of LNxxxx. Depending on the type of \$-language card, the appropriate file type is also assigned to this file. When your Batch job completes successfully, the LNxxxx file is deleted automatically.

The \$-language card has any of the following forms:



/switches

are switches to Batch to tell it how to read your program and whether or not to request a compilation listing when the program is compiled. The switches can be put on the card in any order. The following three switches may be used with any of the \$-language cards. Additional switches are available and can be referenced in the Batch Reference Manual.

/WIDTH:n Switch

Normally, Batch reads up to 80 columns on every card of the source program. You can make Batch stop reading at a specific column by means of the /WIDTH switch. You indicate the number of a column at which to stop. Thus, if you have no useful information in the last 10 columns of each card of your program, you can tell Batch to read only up to column 70 by specifying

/WIDTH:70

ENTERING A BATCH JOB FROM CARDS

/NOLIST Switch

Normally, the \$-language card tells Batch to ask the compiler to generate a compilation listing of your source program. The listing is then printed as part of your job's output. If you do not want this listing, you can include the /NOLIST switch on the \$-language card to stop generation of the listing.

/SUPPRESS Switch

When Batch reads the cards of your source program it normally copies everything on the card up to column 80 or any column you may specify in the /WIDTH switch. However, if you do not want trailing spaces copied (to save space on the disk, for example), you can tell Batch, by means of the /SUPPRESS switch, not to copy any trailing spaces into the disk file.

Examples

The simplest form of the \$-language card is shown in the following example using ALGOL.

`$ALGOL`

This card causes Batch to copy your ALGOL card program into a disk file. The cards in the program are read up to column 80 and trailing spaces are not suppressed. A listing file is produced when the program is compiled. The listing is written as part of the job's output.

The following is an example of a \$ALGOL card with switches.

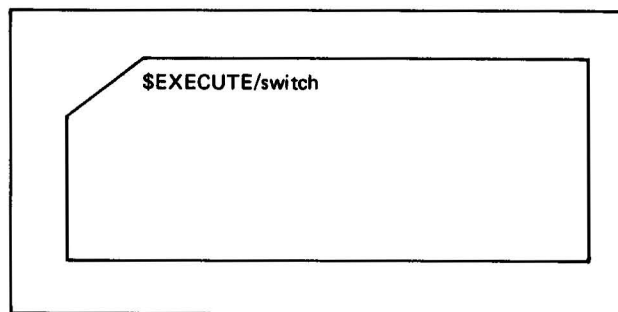
`$ALGOL/NOLIST/SUPPRESS/WIDTH:72`

With this card, Batch copies your program onto disk and inserts a `COMPILE` command into the control file. When the program is compiled, no listing is produced. The cards in the program are read up to column 72, and trailing spaces up to column 72 are not copied into the file.

ENTERING A BATCH JOB FROM CARDS

3.2.6 Executing a Program - The \$EXECUTE Card

The \$EXECUTE card is used to execute the program that has been compiled using the \$-language card. This card is used when the program requires no data or uses data already existing on disk. The form of the \$EXECUTE card is shown below.



/switch is a switch to Batch to tell it what to include in the command it inserts in the control file.

/MAP Switch

If you want a loader map to be generated and printed for you when your program is run, you can specify the /MAP switch on the \$EXECUTE card to tell Batch to request one for you.

An \$EXECUTE card following another \$EXECUTE card in the control file without intervening \$-language cards causes the program executed by the first EXECUTE card to be loaded and executed again.

3.2.7 Executing a Program with Data - The \$DATA Card

The \$DATA card is used when you want to execute a program that uses data from cards. The \$DATA card must be in front of the input data cards. When Batch reads the \$DATA card, Batch copies the data cards that follow it onto a spooled card-reader file and then inserts an EXECUTE command into your control file to execute your program.

When your job is run, any programs are executed that were entered with \$-language cards that came before the \$DATA card. The spooled card reader file becomes the input to the currently executing program, and your program may reference this file by using the card reader as the input device.

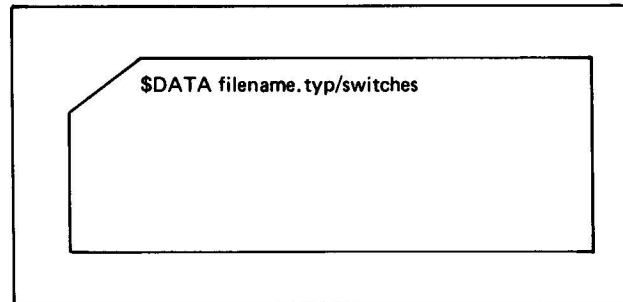
If your input control file contains more than one program and input data, Batch will execute the first program with the input data, spool the results to be printed, and then compile the second program when Batch reads the second \$-language card. Again, when Batch reads the second \$DATA card, a spooled card-reader file is created for your input data cards and an EXECUTE command is inserted in your control file.

A \$DATA card (with its associated card deck) followed by another \$DATA card (with its deck), without intervening \$-language cards, causes the program to be loaded and executed twice. The first deck is used as data on the first execution and the second deck is used as data on the second execution.

ENTERING A BATCH JOB FROM CARDS

If your data is included in the program or is already on disk (so that you do not have cards with data on them), use the \$EXECUTE card (Section 3.2.6) to execute the program.

The form of the \$DATA card is:



filename.typ specifies a name for the input data file (see Section 3.2.7.1). If omitted, a spooled card-reader file is created. You may reference this file by using the card reader as the input device in the source program. If included, a disk file is created, and you may reference this file by using the disk as the input device.

/switches are switches to Batch to tell it how to read your data cards. The switches are described below.

/WIDTH:n Switch

Normally, Batch reads up to 80 columns on every card of your data. You can make Batch stop reading at a specific column by means of the /WIDTH switch, where you indicate the number of a column at which to stop. Thus, if you have no useful information in the last 10 columns of each card of your data, you can tell Batch to read only up to column 70 by specifying

/WIDTH:70

/SUPPRESS Switch

When Batch reads the cards of your data, it normally copies everything on the card up to column 80 or up to any column you may specify on the /WIDTH switch. However, if you do not want trailing spaces copied (to save space on the disk, for example), you can tell Batch, by means of the /SUPPRESS switch, not to copy any trailing spaces into the disk file.

ENTERING A BATCH JOB FROM CARDS

/MAP Switch

If you want a loader map to be generated and printed for you when your program is run, you can specify the /MAP switch on the \$DATA card to tell Batch to request one for you.

Examples

The simplest form of the \$DATA card is:

`$DATA`

This card causes Batch to copy your data into a spooled card-reader file. A spooled card-reader file is a file that Batch creates on disk so that when your program reads from the card reader, that file is read. All 80 columns of the cards are read and trailing spaces are copied into the file.

The following example shows a \$DATA card with switches.

`$DATA MYDAT.DAT/WIDTH:72`

The data that follows this card is copied into a file named MYDAT.DAT and an EXECUTE command is inserted into the control file. When Batch reads the cards of the data, it reads only up to column 72 and copies trailing spaces into the data file.

3.2.7.1 Reading from a Spooled Card-Reader File - If you let Batch assign a name to your data file, you will not know the name that your data file will have; you should, therefore, assign your data file, without a name, to the card reader. The following examples illustrate how to do this.

NOTE

The \$DATA card can be used for data of programs written in ALGOL, COBOL, FORTRAN, and MACRO. It can also be used for programs that are in relocatable binary form. However, data for BASIC programs cannot be copied by means of the \$DATA card because BASIC programs are not compiled and executed. For BASIC programs, use the \$CREATE card as described in Section 3.2.4.

ENTERING A BATCH JOB FROM CARDS

COBOL Example

```
.  
.ENVIRONMENT DIVISION.  
INPUT-OUTPUT SECTION.  
SELECT SALES, ASSIGN TO CDR.  
. .
```

```
.DATA DIVISION.  
FILE SECTION.  
FD SALES, LABEL RECORDS ARE OMITTED.  
. .
```

```
$.DATA (in the control file)
```

FORTRAN Example

To read your data from the card reader, you use the unit number 2 or no unit number, as shown below.

```
.  
.READ (2,f), list  
. .
```

```
END  
$.DATA  
. .
```

```
READ f, list  
. .
```

```
END
```

ENTERING A BATCH JOB FROM CARDS

\$DATA

.
.
.

ALGOL Example

In an ALGOL program, you assign the desired channel (signified by c) to the card reader and select the desired channel. Do not explicitly open the named file on the channel because the file does not have a name that is known to you.

.
.
.
INPUT (c, "CDR")
SELECT INPUT (c)

.
.
.

\$DATA

3.2.7.2 Naming Data Files on the \$DATA Card - If you want to name your data file on the \$DATA card rather than letting Batch name it for you, you must, in your program, assign that file to disk as shown in the following examples.

COBOL Example

.
.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
SELECT SALES, ASSIGN TO DSK.

.
.

.
DATA DIVISION.
FILE SECTION.
FD SALES, VALUE OF IDENTIFICATION IS "SALES CDS".

ENTERING A BATCH JOB FROM CARDS

.
.
.

The \$DATA card would then appear as follows.

```
$DATA SALES.CDS
```

FORTRAN Examples

You can assign your data to disk in several ways when you use FORTRAN. You can read from unit 1, which is the disk, in your program and use the name FOR01.DAT as the filename on your \$DATA card, as shown in the following statements.

```
.  
.  
.  
READ (1,f), list  
.  
.  
.  
$DATA FOR01.DAT
```

You can also tell FORTRAN to read from logical unit 2, which is normally the card reader, and assign unit 2 or the card reader (CDR) to disk (DSK). You can use the name FOR02.DAT on the \$DATA card in this case.

```
.  
.  
.  
OPEN (UNIT=2,DEVICE='DSK')  
READ (2,f), list  
.  
.  
.  
@DEFINE CDR: DSK: (in the control file)  
$DATA FOR02.DAT
```

ALGOL Example

To read your data from the disk in an ALGOL program, you would use the following statements. You can assign your data to any channel (signified by c) and you can give your data file any name as long as the name that you use in your program is the same as that put on the \$DATA card.

ENTERING A BATCH JOB FROM CARDS

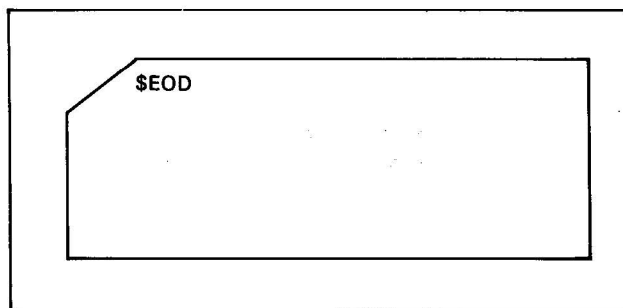
```
.  
.   
.   
INPUT (c, "DSK")  
SELECT INPUT (c)  
OPENFILE (c, "MYDAT.DAT")  
.   
.   
.   
$DATA MYDAT.DAT
```

This is done to ensure that your program finds your data in the disk file under the name that you have assigned to it.

3.2.8 End of Data Input - The \$EOD Card

The \$EOD card terminates the card input that was preceded by either a \$CREATE or \$DATA card.

The form of the \$EOD card is:



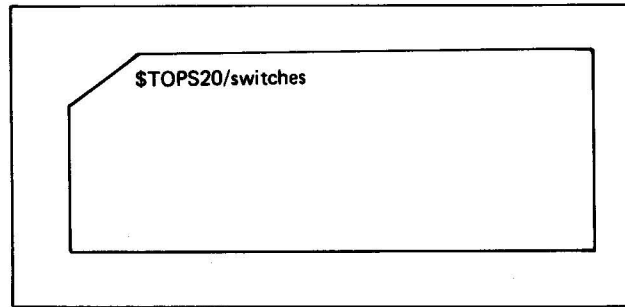
If the \$EOD card does not follow the card input that was preceded by a \$CREATE card, Batch recognizes the next card with a dollar sign (\$) in column one as a new Batch command and as the end of the card input; that is, an EOD card is assumed if one is not present.

3.2.9 System Commands - The \$TOPS20 Card

You can include system commands, commands to system or user programs, and Batch commands in your deck by inserting a \$TOPS20 card immediately before these commands. The \$TOPS20 card directs Batch to copy all cards following it into the Batch control file. Therefore, a single system or Batch command or a group of consecutive system and/or Batch commands must be preceded by a \$TOPS20 card. The copying process is terminated by the next control card in the deck.

ENTERING A BATCH JOB FROM CARDS

The form of the \$TOPS20 card is:



/switches are switches to Batch to tell it how to read and interpret your input.

/WIDTH:n Switch

Normally, Batch reads up to 80 columns on every card of your system or Batch commands. You can make Batch stop reading at a specific column by using the /WIDTH switch, where you indicate the column number at which Batch is to stop reading. Thus, if you have no useful information in the last 10 columns of each card, you can tell Batch to read only up to column 70 by specifying

/WIDTH:70

/SUPPRESS Switch

When Batch reads your cards, it normally copies everything on the card up to column 80 or up to any column you may specify on the /WIDTH switch. However, if you do not want trailing spaces copied (to save space on the disk, for example), you can tell Batch, by means of the /SUPPRESS switch, not to copy any trailing spaces into the disk file.

Figure 3-1 illustrates a sample Batch input card deck using only TOPS-20 commands.

ENTERING A BATCH JOB FROM CARDS

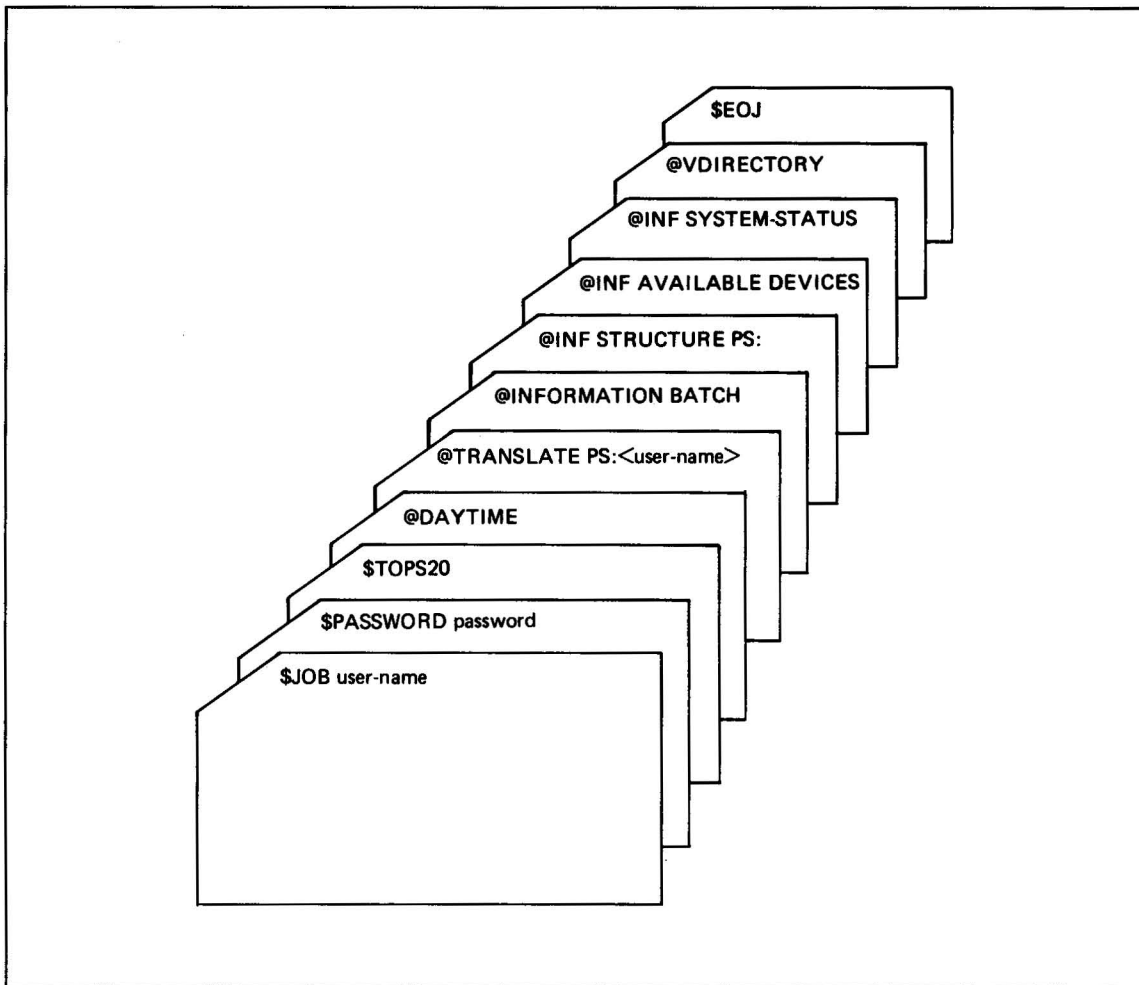


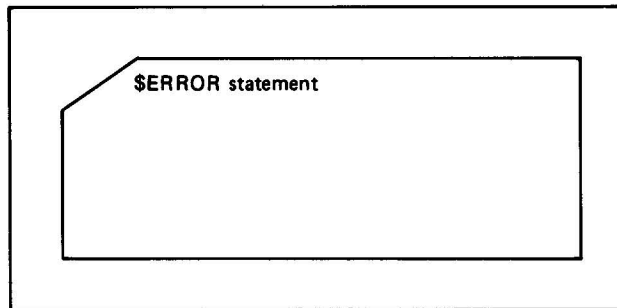
Figure 3-1 Batch Card Deck Using TOPS-20 Commands

3.2.10 Error Recovery - The \$ERROR and \$NOERROR Cards

You can use the \$ERROR card and the \$NOERROR card to recover from errors that may or may not occur while your Batch job is running. If an error occurs during your job (for example, a program fails to compile), Batch will normally terminate your job. However, if a \$ERROR or \$NOERROR card is included in your card deck immediately after the point at which the error occurs, Batch will proceed as indicated on the \$ERROR or \$NOERROR card and will not terminate the job.

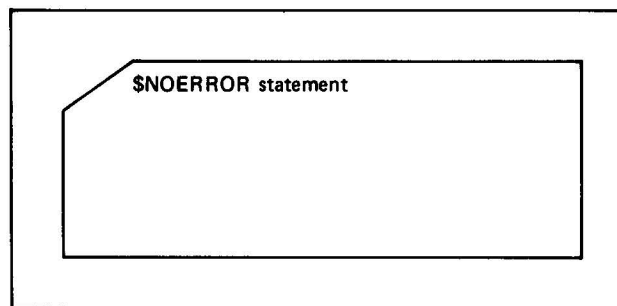
ENTERING A BATCH JOB FROM CARDS

The \$ERROR card has the form:



\$ERROR statement

The \$NOERROR card has the form:



\$NOERROR statement

statement is a system command or a special Batch command (for example, @GOTO or @BACKTO) such as you would include following a \$TOPS20 card. The Batch commands are described in Section 2.3 (of Chapter 2). The statement may also be a comment (begun with the exclamation point(!)) if you wish Batch to simply ignore the error.

If an error occurs in your Batch job and the \$ERROR card is then encountered, the statement on the \$ERROR card is executed and the job continues. If the \$ERROR card is encountered when an error has not occurred, the card is ignored.

If an error occurs in your Batch job and the \$NOERROR card is encountered, no action will be taken, with the exception that Batch will not terminate your job as it would have if the card had not been found. If the \$NOERROR card is encountered when no error has occurred, the statement on the \$NOERROR card is executed.

The \$ERROR card is equivalent to the Batch command @IF(ERROR). The \$NOERROR card is equivalent to the Batch command @IF(NOERROR). See Section 3.5 for examples of Batch jobs using the \$ERROR and \$NOERROR cards.

ENTERING A BATCH JOB FROM CARDS

3.3 SETTING UP YOUR CARD DECK

Batch enters commands into the control file when you use certain control cards. Where you put these control cards in your card deck determines their position in the control file. Batch reads your card deck in sequential order, copying commands into the control file as they or the special control cards are read. However, when Batch reads a control card that tells it to copy a program or data into a disk file, the disk file is created immediately, before the remainder of the job is processed. Every succeeding card is copied until another control card is read.

A Batch job can do almost anything a timesharing job can do. If you wish to perform complicated tasks, you may include system commands in your deck to direct Batch to execute these tasks. Section 3.4 describes the way to include system commands for the desired control.

The \$JOB card, the \$PASSWORD card, and the \$EOJ card are required for all jobs. The \$JOB card must be the first card in the deck and must be immediately followed by the \$PASSWORD card. The \$EOJ card must be the last card in the deck.

The control cards used to compile and execute programs written in ALGOL, COBOL, FORTRAN, and MACRO are shown in Figure 3-2. The following card deck does not apply to control card decks for BASIC. Refer to Section 3.4.1 for information regarding BASIC.

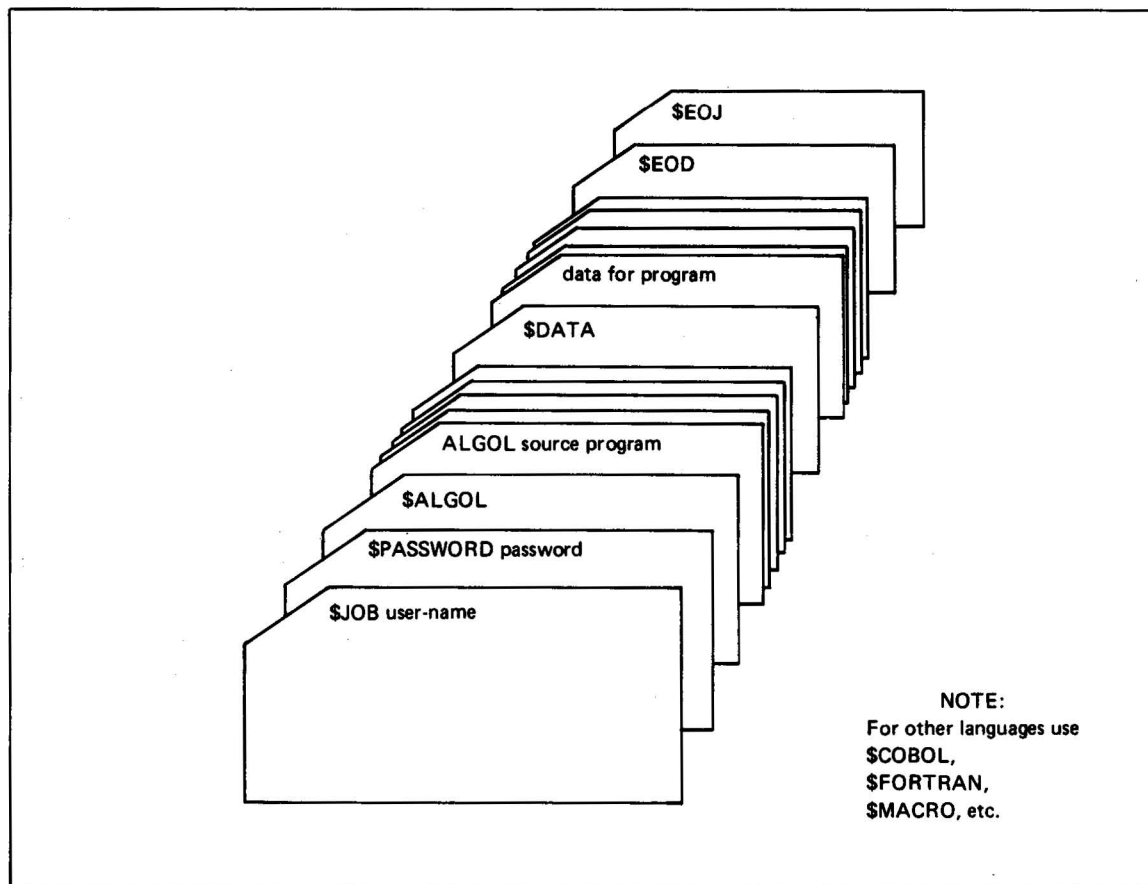


Figure 3-2 Typical Program Card Deck

ENTERING A BATCH JOB FROM CARDS

The typical card deck shown in Figure 3-2 includes a language card (\$ALGOL, \$COBOL, etc.) immediately prior to the source program. This language card informs Batch of the system program to be employed for processing (compiling) the succeeding cards. The \$DATA card likewise immediately precedes the data cards to inform Batch that the succeeding cards contain data for the program. In both cases, the information is stored (on a spooled card-reader file) to establish program files and data files. The \$DATA card also causes Batch to execute the program, using the data cards as input. The \$EOJ card informs Batch that all cards pertaining to the job have been entered. At this time Batch has access to the program to be compiled and the data to be used by the program; it knows what compiler or assembler is to be used, and has built a control file containing the TOPS-20 @EXECUTE command so that the program will be run.

3.4 PUTTING COMMANDS INTO THE CONTROL FILE FROM CARDS

Batch enters commands into the control file when you use certain control cards such as \$EXECUTE and \$DATA. However, only a small number of operations, such as compilation and execution of programs, can be put into the control file using control cards. To perform operations in your control file other than compilation or execution, you must include commands in your card deck for Batch to copy into your control file. If you want to include Batch commands or system commands in your card deck, you must insert a \$TOPS20 card immediately before these commands in your deck. The \$TOPS20 card directs Batch to copy all succeeding commands into the control file until the next control card is encountered. The commands will later be executed by Batch in the same order that they appear in your card deck.

For example, in order to compare two card decks and produce a list of the differences, you could include the cards shown in Figure 3-3 in your deck.

ENTERING A BATCH JOB FROM CARDS

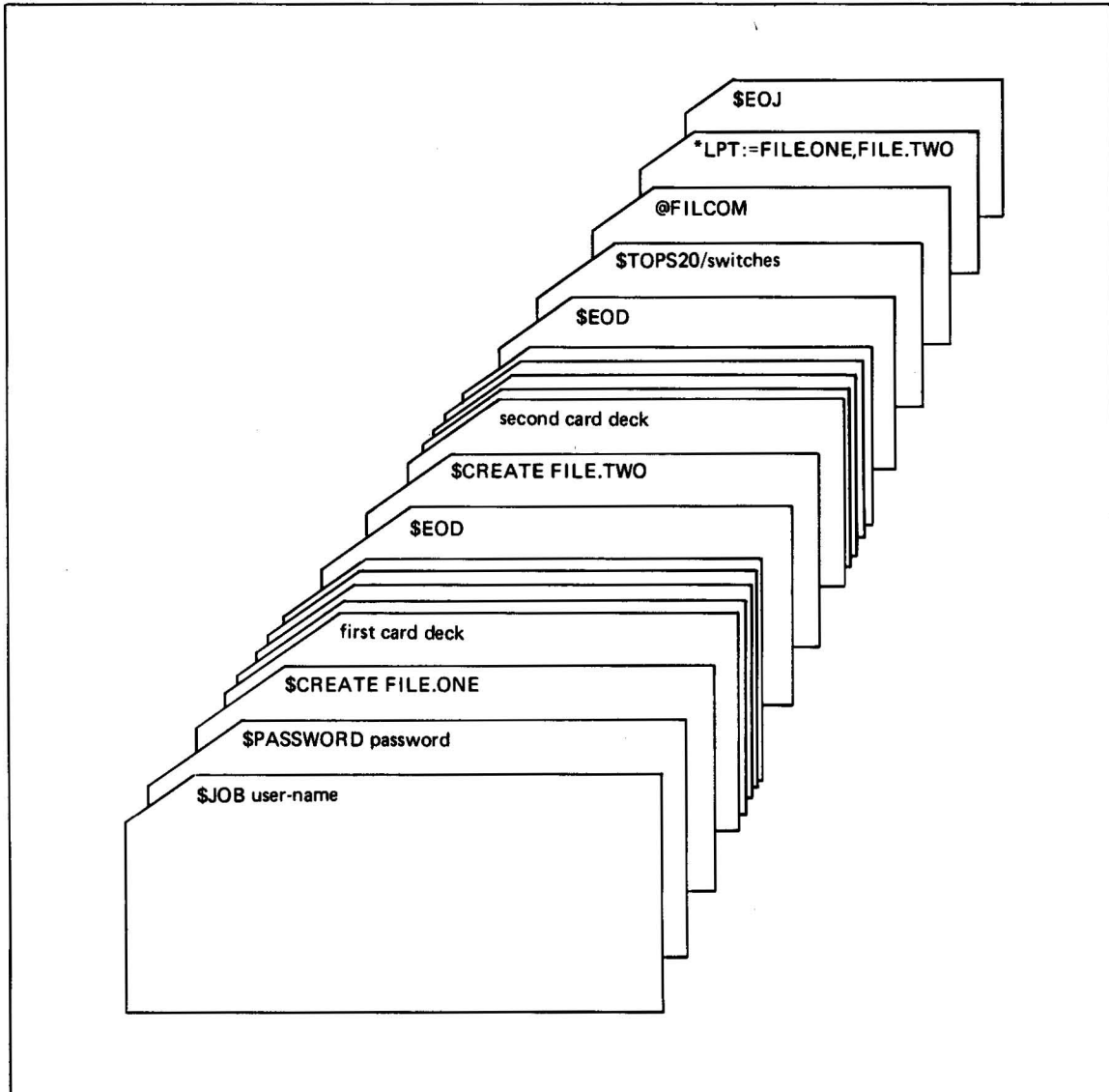


Figure 3-3 Use of Control Cards to Compare Two Card Decks

The only system commands that you cannot use in a Batch job are ATTACH and SET TIME-LIMIT. Batch will send these commands to the EXEC, the EXEC will give an error, and Batch will detect the error and terminate your job. Also, you cannot use the LOGIN command in your Batch job because you will get an error that will terminate your job. Batch logs in your job in accordance with your \$JOB and \$PASSWORD cards.

3.4.1 Card Decks for Programs that Do Not Have Special Control Cards

By using system commands and the \$CREATE control card, you can process any program that does not have special control cards. You put a \$CREATE card in front of a program, data, or any other group of cards to make Batch copy the cards into a disk file and, if you request, to

ENTERING A BATCH JOB FROM CARDS

print the file on the line printer. The \$CREATE card is described in detail in Section 3.2.4. You put the \$TOPS20 card in front of monitor and Batch commands to cause Batch to copy these commands into the control file. The \$TOPS20 card is described in detail in Section 3.2.9.

For example, a BASIC program does not have a specific control card. To run a BASIC program under Batch from cards, you can combine the \$CREATE card and the \$TOPS20 card with system commands. You can also use a \$CREATE card to copy the data which a BASIC program will use. The \$DATA card cannot be used, because the \$DATA card puts an EXECUTE command into the control file, and BASIC does not use the EXECUTE command to run. The \$TOPS20 card causes Batch to copy the monitor commands into the control file.

Figure 3-4 shows a card deck that enters a BASIC program for running under Batch.

ENTERING A BATCH JOB FROM CARDS

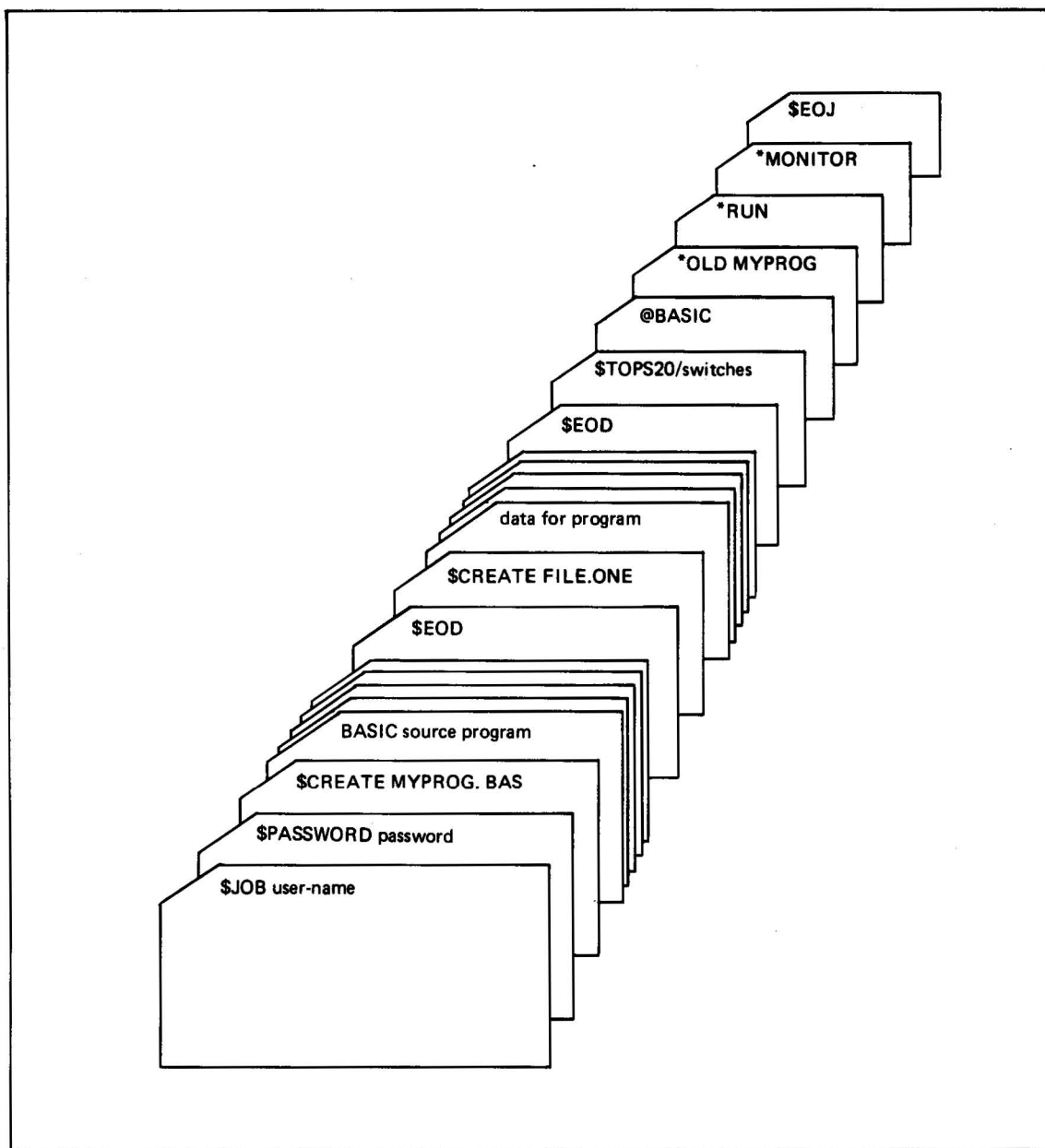


Figure 3-4 BASIC Program Card Deck with Integral Data

The BASIC program contains statements that read data from disk file FILE.ONE. You answer OLD to the BASIC question

NEW OR OLD-

because the program file is on disk and can be retrieved by BASIC.

If your BASIC program reads data that is to be input from a terminal during the running of the program, enter the data in the control file so that it will be passed to your program by Batch. This is shown in Figure 3-5.

ENTERING A BATCH JOB FROM CARDS

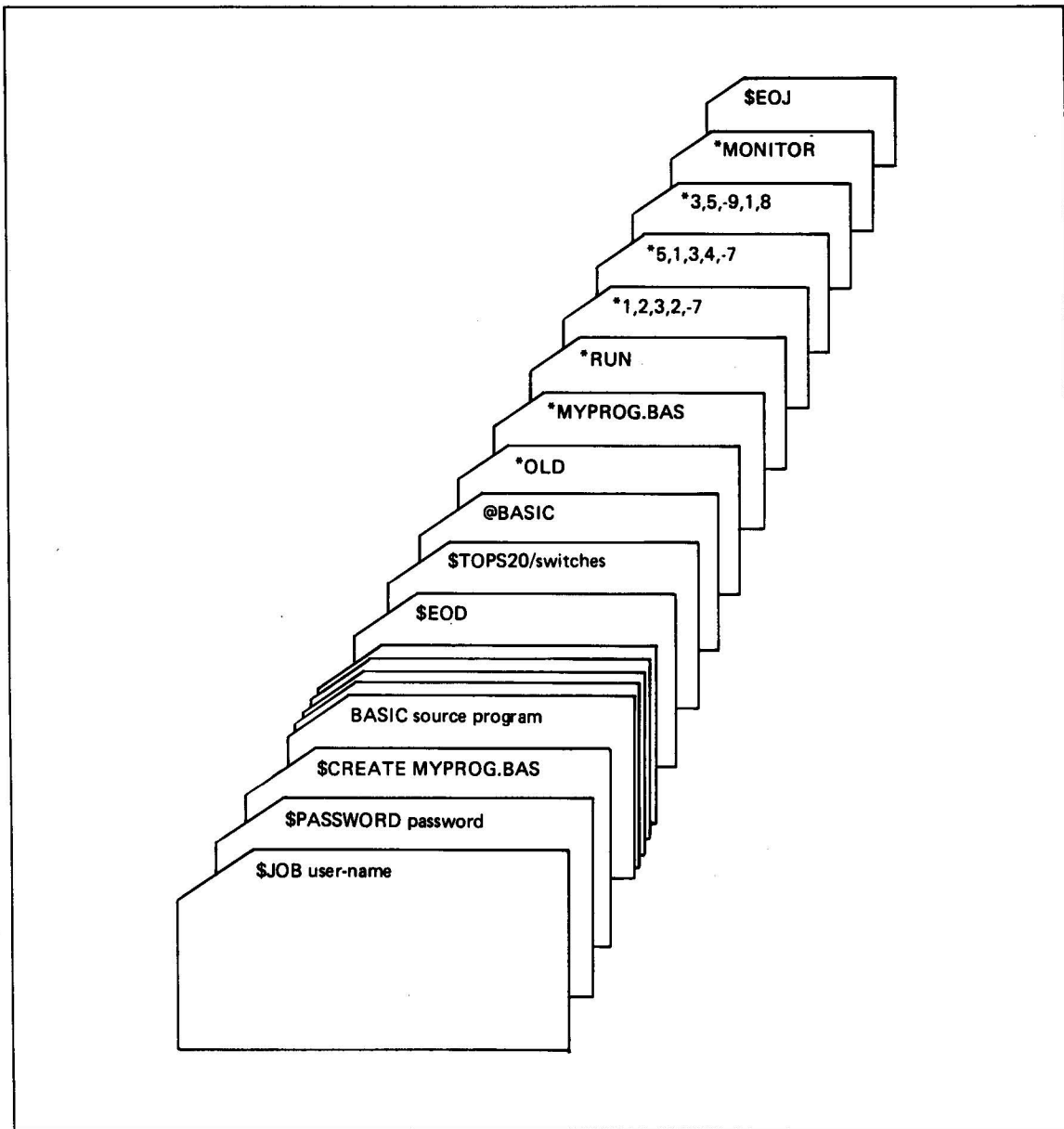


Figure 3-5 BASIC Program Card Deck with Provisions for Terminal Data Input

You can use the same technique to enter programs written in any language that does not have a specific control card provided that your installation supports the language. Also, you can run system programs under Batch using the same technique.

ENTERING A BATCH JOB FROM CARDS

3.5 SPECIFYING ERROR RECOVERY IN THE CONTROL FILE

Normally, when an error occurs in your job, Batch terminates the job. However, you can specify recovery from errors in the control file by means of the \$ERROR and \$NOERROR cards, described in Section 3.2.10. You must include one of these cards at the point in the control file where an error may occur. When an error occurs, Batch examines the next system command level line (skipping over lines that contain data or command strings of a system program) to find an @IF (ERROR) statement or @IF(NOERROR) statement to tell it what to do about the error. If an error does not occur and an @IF (ERROR) statement is present, the @IF (ERROR) statement is ignored. If an error occurs and an @IF(NOERROR) statement is present, the statement is ignored with the exception that Batch does not terminate the job.

Thus, if you have a program that you are not sure is error free, you can include a \$ERROR or \$NOERROR card to tell Batch what to do if an error occurs, as shown in Figure 3-6.

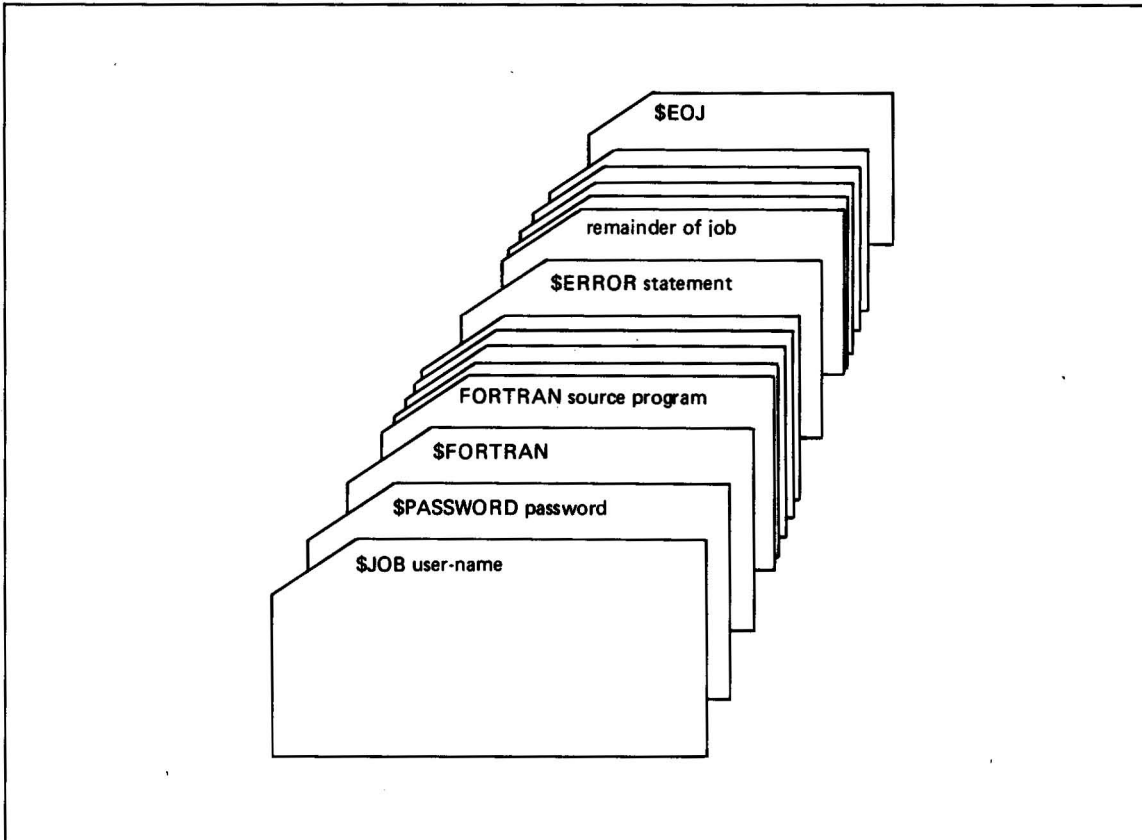


Figure 3-6 Card Deck with Error Statement

The above cards would cause Batch to make the following entries in the control file.

```
@COMPILE . . .  
@IF (ERROR) statement
```

ENTERING A BATCH JOB FROM CARDS

On either the \$ERROR or \$NOERROR card, you must include a statement that tells Batch what to do. You can use any system command, a command to a program, or one of the special Batch commands. The @GOTO and @BACKTO Batch commands are commonly used for this purpose. Refer to Sections 2.3.4 and 2.3.5 for descriptions of these commands. If you use @GOTO or @BACKTO on your \$ERROR or \$NOERROR card, be sure that you supply a line for the control file that has the label you specified in the @GOTO or @BACKTO command.

Two sample jobs are shown on the following pages. The first shows the use of the \$ERROR card and the @GOTO command to specify error recovery. The second example shows the use of the \$NOERROR card and the @GOTO command.

If you have a program that may compile with errors, you can include another version of the same program in your job (that hopefully will compile) and tell Batch to compile the second program if the first has an error. The cards to enter this job are shown in Figure 3-7.

ENTERING A BATCH JOB FROM CARDS

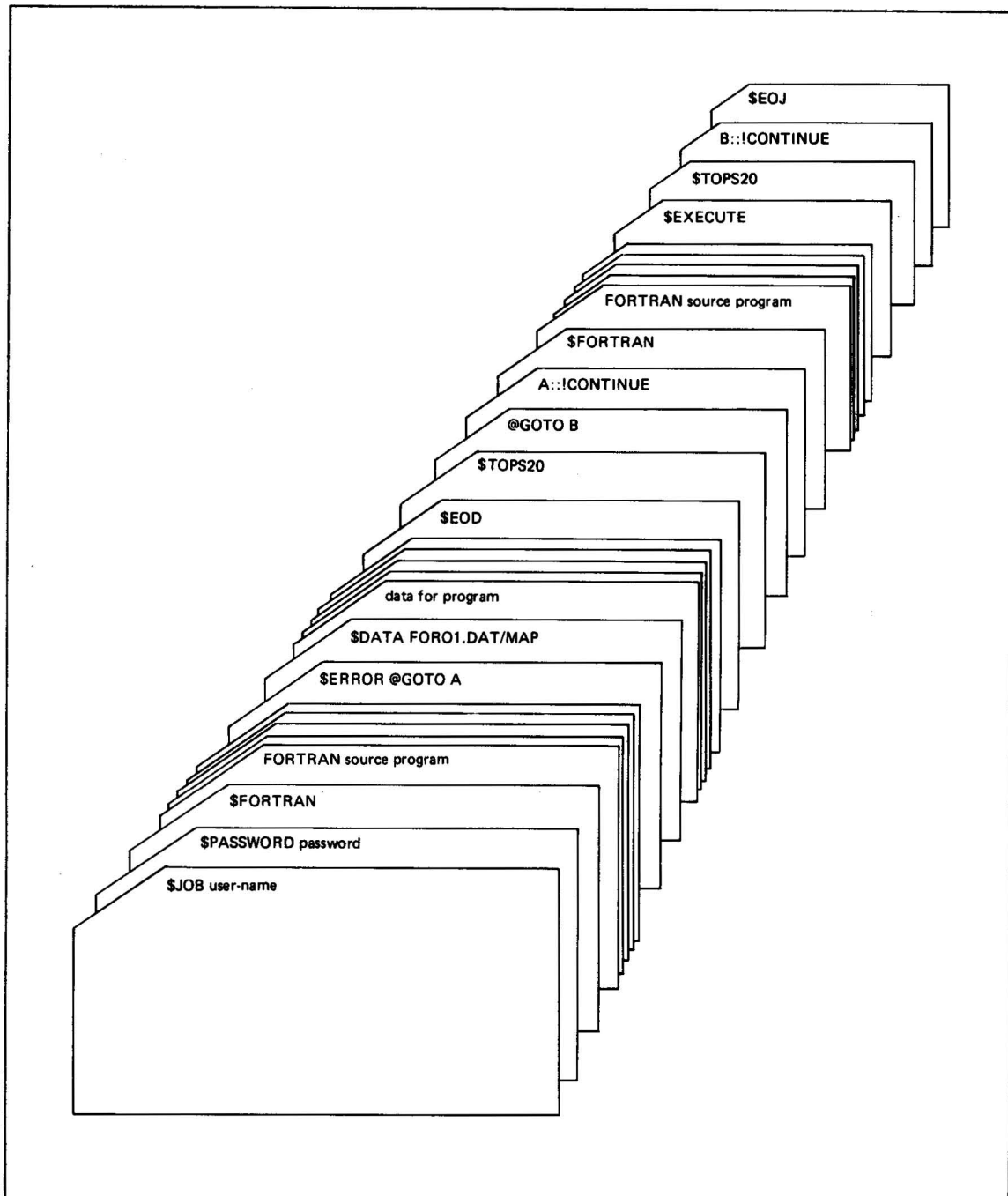


Figure 3-7 Card Deck with Error Recovery Program

ENTERING A BATCH JOB FROM CARDS

These cards set up the following control file for you.

```
@COMPILE/COMPILE LNxxxx.FOR/LIST
@IF (ERROR) @GOTO A
@EXECUTE LNxxxx.REL/MAP:MAP.LST
@GOTO B
A::!CONTINUE
@COMPILE/COMPILE LNxxxx.FOR/LIST
@EXECUTE LNxxxx.FOR
B::!CONTINUE
```

The \$FORTRAN card told Batch to copy the program into a disk file, to create a unique filename for the program in the form LNxxxx.FOR, and to insert a COMPILE command into the control file. The \$ERROR card told Batch to insert @IF (ERROR) @GOTO A into the control file. The data was copied into a disk file and an EXECUTE command was put into the control file because of the \$DATA card. The \$TOPS20 card told Batch to start copying cards into the control file, so Batch put the next two lines into the control file. The second \$FORTRAN card told Batch to copy the program into a disk file, create another unique filename for the program in the form LNxxxx.FOR, and put a COMPILE command into the control file. A \$EXECUTE card was used instead of a \$DATA card because the data for the second program was already in a file on disk. The \$TOPS20 card caused the next line to be put into the control file.

When the job is started, Batch reads the control file and passes commands to the system. If an error occurs in the compilation of the first program, Batch executes the @GOTO command within the @IF statement. The command tells Batch to skip to the line labeled A, which contains a comment. Batch then proceeds to the next line. The second program is compiled and executed with the data. The next line is a comment, so Batch continues to the end of the control file. If an error does not occur in the first program, Batch skips the @IF statement, executes the program with the data, avoids the second program by skipping to label B, and continues to the end of the control file.

A variation of the above procedure using the \$NOERROR card and @GOTO command is shown in Figure 3-8. The difference is that Batch skips the @IF statement if an error occurs and performs it if an error does not occur.

ENTERING A BATCH JOB FROM CARDS

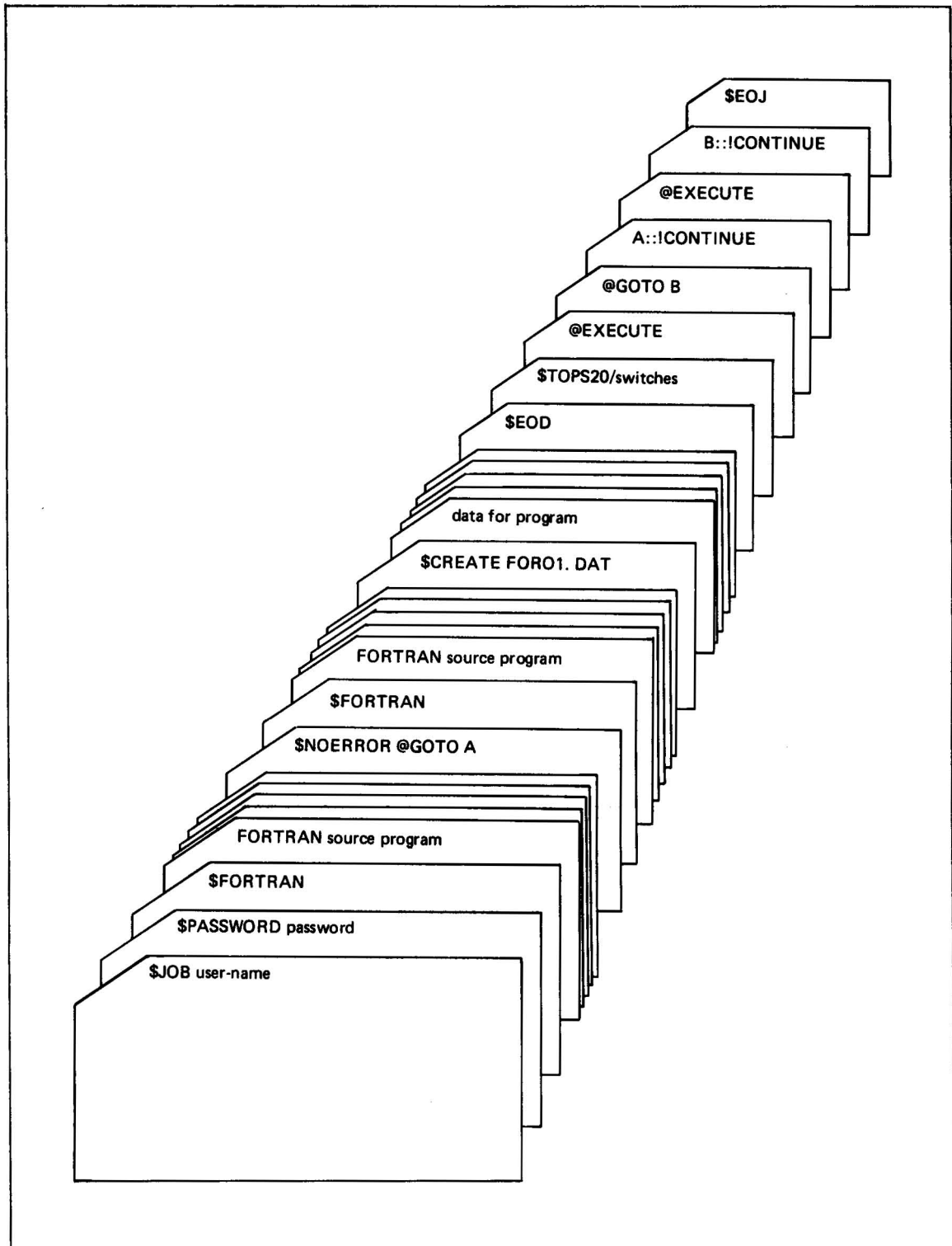


Figure 3-8 Card Deck Using GOTO Statement

ENTERING A BATCH JOB FROM CARDS

Batch reads the cards and puts the following commands into the control file.

```
@COMPILE/COMPILE LNxxxx.FOR/LIST
@IF (NOERROR) @GOTO A
@COMPILE/COMPILE LNxxxx.FOR/LIST
@EXECUTE LNxxxx.FOR
@GOTO B
A:;!CONTINUE
@EXECUTE LNxxxx.FOR
B:;!CONTINUE
```

The \$FORTRAN card tells Batch to copy the FORTRAN program into a file, to create a unique filename of the form LNxxxx.FOR, and to insert a COMPILE command into the control file. The \$NOERROR card tells Batch to insert an @IF command into the control file.

The second \$FORTRAN card tells Batch to copy the second program into a disk file, to create a unique filename of the form LNxxxx.FOR, and to insert another COMPILE command into the control file. Instead of a \$DATA card, a \$CREATE card is used to tell Batch to copy the data into a disk file named FOR01.DAT. The \$DATA card is not used here because it would have the names of both programs in its list for the EXECUTE command generation, which would cause an error when the job is run. To tell Batch to start copying cards into the control file, the \$TOPS20 card comes next. Thus, Batch copies the next five cards into the control file.

When the job is run, Batch passes the COMPILE command to the system to compile the first program. If an error does not occur, the @IF command is read and the @GOTO command is executed. Batch skips to the line labeled A, which is a comment, and continues reading the control file. The program LNxxxx.FOR is executed with the data, and the end of the job is reached. If an error occurs, Batch skips the @IF statement and continues reading the control file. The second program is compiled and then executed with the data. Batch is then told to go to the line labeled B, which is a comment line. The end of the job follows. The TOPS-20 @EXECUTE command was used in this job rather than the \$EXECUTE card. The \$EXECUTE card would have caused the names of both programs to be included in the @EXECUTE command which would have resulted in an error when the job was run.

The examples shown above illustrate only two ways that you can specify error recovery in the control file. You can use the @BACKTO command or any system command that you choose to help you recover from errors in your job.

However, you do not have to attempt to recover from errors while your job is running. You can correct your errors according to the error messages in the log file when your job is returned to you, and then run your job again.

CHAPTER 4

INTERPRETING YOUR PRINTED OUTPUT

You can receive three kinds of printed output from your Batch jobs:

1. Output that you request, i.e., the results of your job.
2. Output from Batch commands, i.e., the log file.
3. Output that is the result of actions by your job, Batch, the system, or system programs; examples of this output are compilation listings, cross-reference listings, and error messages.

4.1 OUTPUT FROM YOUR JOB

If your job uses the PRINT command to print files on the line printer, the files will be printed in listings separate from the log file. The printed output from each program will be preceded by two banner pages containing your user name and other pertinent information. Following these pages are two header pages that contain the name of your output file in block letters; the output follows these header pages. Two trailer pages follow your output; they contain the same information that is on the first two pages. The header and trailer pages also include three rows of numbers (read vertically from 001 to 132) that represent the character print positions on the line printer.

If your output is usually directed to the terminal, it will be printed in the log file, not as a separate file. In the sample output shown in Section 4.4, the output from the program is included in the log file because it was directed to the terminal rather than the line printer.

Although this chapter deals mainly with printed output, you can have output to any device that the installation supports, as long as the installation allows you to use these devices.

4.2 BATCH OUTPUT

The output from Batch consists of a log file that contains all the statements in the control file, commands sent to the system from Batch for you, and the replies to the commands from the system and from system programs like the compilers. Any error message sent from the system or system program or from Batch itself is also written in the log file.

INTERPRETING YOUR PRINTED OUTPUT

When your Batch card job is printed on the line printer, it will have a unique filename of JBxxxx and a file type of .LOG. This file is deleted automatically after it has printed.

4.3 OTHER PRINTED OUTPUT

Other output that you can get as a result of action by your job includes compiler listings, cross-reference listings, and loader maps for programs that were successfully loaded.

The compiler and cross-reference listings are those listings generated by the compiler if you request them. When you enter your job from cards, Batch requests compilation listing for you unless you specify otherwise. Cross-reference listings are generated for you only if you specifically ask Batch for them. When you enter your job from a terminal, you must request the listings in the COMPILE command. Refer to the Batch Reference Manual for the switches (/CREF, /MAP) that are available to generate additional listings for your Batch job process.

If a fatal error occurs in a program in your job and you have not included an error recovery command to Batch, Batch will not try to recover from the error for you. Instead, it will write the error message in the log file and terminate your job.

4.4 SAMPLE BATCH OUTPUT

Two sample jobs and their output are shown in the following sections. The first shows a job entered from a terminal, the second shows a job entered from cards. The log file is somewhat different for the two types of jobs.

4.4.1 Sample Output of a Job from a Terminal

The following example illustrates a job as it would be entered from a terminal. You would first create the program as a file on disk.

```

      ESC
      ↓
@CREATE (FILE) COBOL1.CBL RET
Input: COBOL1.CBL.1
00100 IDENTIFICATION DIVISION. RET
00200 PROGRAM-ID. COBOL1. RET
00300 ENVIRONMENT DIVISION. RET
00400 DATA DIVISION. RET
00500 PROCEDURE DIVISION. RET
00600 START. RET
00700 DISPLAY "THIS IS TO SHOW SAMPLE OUTPUT FROM BATCH." RET
00800 DISPLAY "THESE TWO LINES ARE OUTPUT FROM THE PROGRAM." RET
00900 STOP RUN. RET
01000 $
      ↑
      ESC
*E RET

[[COBOL1.CBL.1]]
@
```

INTERPRETING YOUR PRINTED OUTPUT

Then you would make up a control file to compile and execute the COBOL program.

```
      ESC
      ↓
@CREATE (FILE) MYJOB.CTL RET
Input: MYJOB.CTL.1
00100  @COMPILE COBOL1.CBL/LIST RET
00200  @EXECUTE COBOL1.CBL RET
00300  $
      ↑
      ESC
      ↓
*E RET

[MYJOB.CTL.1]
@
```

You would then submit the job to Batch using the SUBMIT command.

```
      ESC
      ↓
@SUBMIT (BATCH JOB) MYJOB.CTL RET
[INP:MYJOB=/Seq:4453/Time:0:05:00]
@
```

When the job is run, the program is compiled and a listing is produced. The following listing is placed in the queue of the line-printer spooler:

```
PROGRAM COBOL1 COBOL-68 12(526) BIS 5-APR-78 14:42
COBOL1.CBL 05-APR-78 14:52

0001 00100 IDENTIFICATION DIVISION,
0002 00200 PROGRAM-ID, MYPROG,
0003 00300 ENVIRONMENT DIVISION,
0004 00400 DATA DIVISION,
0005 00500 PROCEDURE DIVISION,
0006 00600 START,
0007 00700 DISPLAY "THIS IS TO SHOW SAMPLE OUTPUT FROM BATCH,",
0008 00800 DISPLAY "THESE TWO LINES ARE OUTPUT FROM THE PROGRAM,",
0009 00900 STOP RUN,

NO ERRORS DETECTED
```

The log file below is printed as your job's output. The output from the program is written in the log file because it is output to the terminal and the log file simulates the terminal dialogue. The log file also contains some commands that Batch sent to the system for you and some additional system information. An annotated log file is shown on the following page. Note that each line in the log file is preceded by the time of day when the line was written. Following the time is a word that describes what kind of information is on each line. Much of the information is system information and is described in detail in the DECSYSTEM-20 (TOPS-20) Batch Reference Manual.

INTERPRETING YOUR PRINTED OUTPUT

```

17:49:09 BAJOB  BATCON version 103(3000) running MyJOB sequence 4453 in stream 1
17:49:09 BAFIL  Input from PS:<USER-NAME>MYJOB,CTL,1
17:49:09 BAFIL  Output to PS:<USER-NAME>MYJOB,LOG
17:49:09 BASUM  Job parameters
                  Time:00:05:00 Unique:YES Restart:NO Output:LOG

17:49:09 MONTR  SYSTEM 2102 DEVELOPMENT SYSTEM, TOPS-20 Monitor 3A(1475)
17:49:10 MONTR  $LOGIN USER-NAME 341
17:49:10 MONTR  Job 6 on TTY225 5-Apr-78 17:49:13
17:49:14 MONTR  @
17:49:14 MONTR  [CONNECTED TO PS<USER-NAME>]
17:49:14 MONTR  @SET TIME=LIMIT 300
17:49:15 MONTR  @@COMPILE COBOL1,CBL/LIST
17:49:18 USER   COBOL1 COBOL1 [COBOL1,CBL]
17:49:32 USER   EXIT
17:49:32 USER   @@EXECUTE COBOL1,CBL
17:49:33 USER   LINK; Loading
17:49:44 USER   [LNKXCT COBOL1 Execution]
17:49:45 USER   THIS IS TO SHOW SAMPLE OUTPUT FROM BATCH,
17:49:45 USER   THESE TWO LINES ARE OUTPUT FROM THE PROGRAM,
17:49:45 USER   EXIT
17:49:45 USER   @nC
17:49:45 MONTR  $LOGOUT
17:49:47 MONTR  Killed Job 6, User USER-NAME, Account 341, TTY 225,
17:49:47 MONTR  at 5-Apr-78 17:49:47, Used 01013 in 010133
17:49:49 LPDAT  [LPTLSJ LPTSPL version 103(2305) running on PLPT1, 5-Apr-78]
17:49:49 LPDAT  [LPTSJS Starting Job MYJOB, Seq #4453, request created at 5-Apr-78]
17:49:50 LPMSC  [LPTSTF Starting File PS:<SPOOL>LPT-226-0-COBOL1,LST,12]
17:50:05 LPMSC  [LPTFPF Finished Printing File PS:<SPOOL>LPT-226-0-COBOL1,LST,12]

```

INTERPRETING YOUR PRINTED OUTPUT

4.4.2 Sample Output of a Job on Cards

This example shows a job in which a small COBOL program is compiled and executed. The card deck is shown in Figure 4-1.

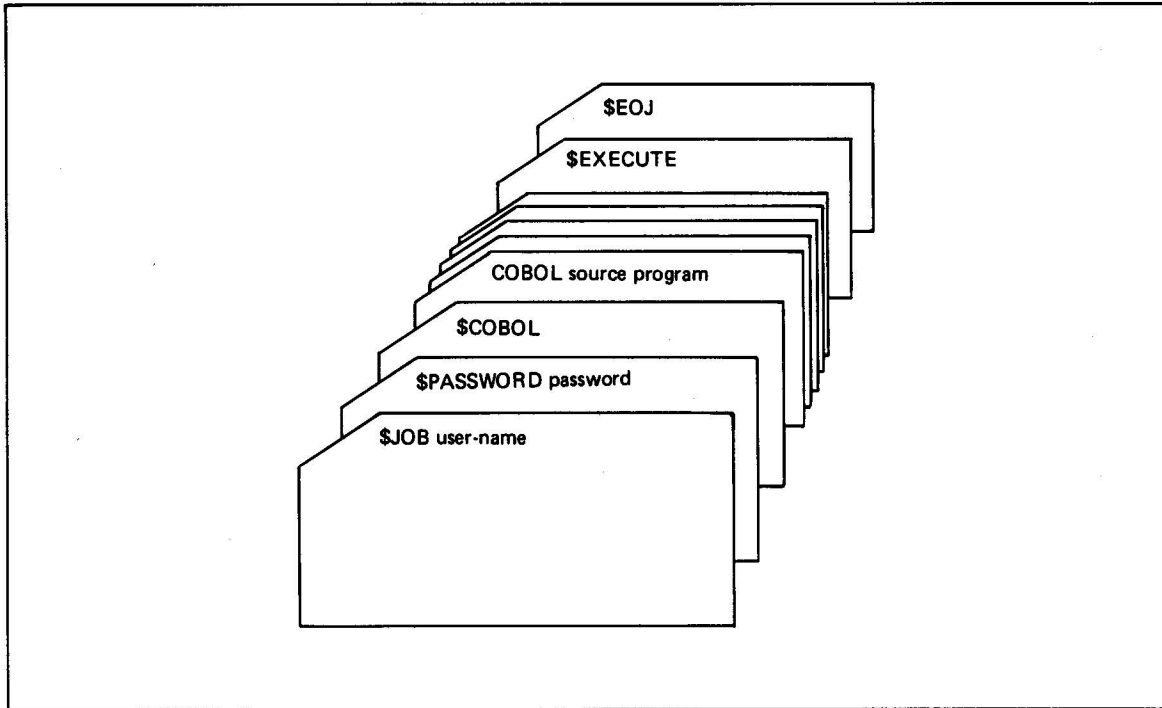


Figure 4-1 COBOL Print Program Card Deck

The COBOL program is as follows.

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. COBOL1.  
ENVIRONMENT DIVISION.  
DATA DIVISION.  
PROCEDURE DIVISION.  
START.  
DISPLAY "THIS IS TO SHOW SAMPLE OUTPUT FROM BATCH."  
DISPLAY "THESE TWO LINES ARE OUTPUT FROM THE PROGRAM."  
STOP RUN.
```

When the job is run, the program is compiled and a compilation listing is produced. The listing is shown below. Note that the compiler puts sequence numbers on the program even though they were not in the original program.

INTERPRETING YOUR PRINTED OUTPUT

PROGRAM COBOL1 COBOL-60 12(526) BIS 7-APR-78 07101
LN2REC,CBL 07-APR-78 07:01

```
0001  IDENTIFICATION DIVISION,  
0002  PROGRAM=ID, COBOL1,  
0003  ENVIRONMENT DIVISION,  
0004  DATA DIVISION,  
0005  PROCEDURE DIVISION,  
0006  START,  
0007  DISPLAY "THIS IS TO SHOW SAMPLE OUTPUT FROM BATCH,"  
0008  DISPLAY "THESE TWO LINES ARE OUTPUT FROM THE PROGRAM,"  
0009  STOP RUN,
```

NO ERRORS DETECTED

After the program is compiled, it is loaded and executed. The program in this example does not have output to the line printer. Instead its output is written to a terminal. Because this is a Batch job, the terminal output is written in the log file. The log file is printed because the end of the job is reached. The log file contains all the dialogue between your job and the system and system programs, and some commands that Batch sent to the system for you. An annotated log file is shown on the following pages. Note that each line in the log file is preceded by the time of day when the line was written. Following the time is a word that describes what kind of information is on each line. Refer to the DECSYSTEM-20 (TOPS-20) Batch Reference Manual for a description and definition of these words.

INTERPRETING YOUR PRINTED OUTPUT

```

07:01:46 STDAT 7-APR-78 SYSTEM 2102 DEVELOPMENT SYSTEM, TOPS-20 Monitor 3A(1475)
07:01:46 STCRD $JOB USER=NAME /ACCOUNT:341
07:01:46 STCRD $COBOL
07:01:47 STMSG File PS:LN2REC,CBL Created = 9 Cards Read
07:01:47 STCRD $EXECUTE
07:01:47 STCRD $EOJ
07:01:47 STSUM End of Job Encountered
07:01:47 STSUM 14 Cards Read
07:01:47 STSUM Batch Input Request Created

07:01:54 BAJOB BATCON version 103(3000) Running JB2RED sequence 4301 in stream 1
07:01:54 BAFIL Input from PS:<USER=NAME>JB2RED,CTL,1
07:01:54 BAFIL Output to PS:<USER=NAME>JB2RED,LOG,1
07:01:54 BASUM Job parameters
Time:00:03:00 Unique:YES Restart:YES Output:LOG

07:01:54 MONTR
07:01:54 MONTR SYSTEM 2102 DEVELOPMENT SYSTEM, TOPS-20 Monitor 3A(1475)
07:01:54 MONTR $LOGIN USER=NAME 341
07:01:57 MONTR Job 38 on TTY221 7-APR-78 07:01:57
07:01:57 MONTR 0
07:01:57 MONTR [CONNECTED TO PS:<USER=NAME>]
07:01:57 MONTR $SET TIME-LIMIT 300
07:01:57 MONTR $COMPILE /COMP/COB PS:LN2REC,CBL/LIST
07:01:57 MONTR COBOL: COBOL: [LN2REC,CBL]
07:02:05 USER
07:02:05 USER EXIT
07:02:05 MONTR $EXECUTE /REL PS:LN2REC,REL
07:02:06 USER LINK: Loading
07:02:10 USER [LNKXCT COBOL: Execution]
07:02:11 USER THIS IS TO SHOW SAMPLE OUTPUT FROM BATCH.
07:02:11 USER THESE TWO LINES ARE OUTPUT FROM THE PROGRAM.
07:02:11 USER
07:02:11 USER EXIT
07:02:11 MONTR 0
07:02:11 BLABL $ERR:1
07:02:11 MONTR $FIN:1
07:02:11 MONTR $DELETE PS:LN2REC,CBL,PS:LN2REC,REL
07:02:11 MONTR <USER=NAME>LN2REC,CBL,1 (OK)
07:02:11 MONTR <USER=NAME>LN2REC,REL,1 (OK)
07:02:11 MONTR 0
07:02:11 MONTR $LOGOUT
07:02:13 MONTR Killed Job 38, User USER=NAME, Account 341, TTY 221,
07:02:13 MONTR at 7-APR-78 07:02:13, Used 0:0:3 in 0:0:15
07:02:14 LPDAT [LPTLSJ LPT8PL version 103(2305) running on PLPT0, 7-APR-78]
07:02:14 LPDAT [LPTJSJ Starting Job JB2RED, Seq #4301, Request created at 7-APR-78]
07:02:14 LPMSG [LPTSTF Starting File PS:<SPOOL>LPT-226-0-LN2REC,LST,16]
07:02:30 LPMMSG [LPTFPF Finished printing File PS:<SPOOL>LPT-226-0-LN2REC,LST,16]

```


CHAPTER 5

EXAMPLES OF COMMON TASKS WITH BATCH

This chapter shows some sample jobs that are run from a terminal and from cards. Section 5.1 illustrates entering jobs from a terminal. Section 5.2 shows entering jobs from cards. The examples are the same in both cases, the difference is in the way that they are entered.

5.1 USING THE TERMINAL TO ENTER JOBS

ALGOL Example

The first job is a simple ALGOL program that writes output to the terminal. Since the job is being entered through Batch, the output is written in the log file instead of on an actual terminal.

```

      ESC
      ↓
@CREATE (FILE) MYPROG.ALG RET
INPUT: MYPROG.ALG.1
00100 BEGIN RET
00200 REAL X;INTEGER I; RET
00300 X:=1; RET
00400 FOR I:= 1 UNTIL 1000 DO X := X+I ; RET
00500 PRINT (X); RET
00600 END RET
00700
      ESC
      ↓
*E RET

[MYPROG.ALG.1]
@
```

The control file for the program is created as follows.

```

      ESC
      ↓
@CREATE (FILE) MYFILE RET
INPUT: MYFILE..1
00100 @COMPILE MYPROGRAM.ALG/LIST RET
00200 @EXECUTE MYPROG.ALG RET
00300
      ESC
      ↓
*E RET

[MYFILE..1]
@
```

EXAMPLES OF COMMON TASKS WITH BATCH

To execute this ALGOL program using the Batch control file, issue the SUBMIT command.

```

      ESC
      ↓
@SUBMIT (BATCH JOB) MYFILE.. RET
CINP:MYFILE=/SEQ:4067/TIME:0:05:00]
@
  
```

When Batch starts the job, the statements in the control file call the ALGOL compiler to compile the program. The TOPS-20 EXEC then calls the loader to load the program for execution. A listing of the program will be printed with the log file shown below.

```

09:59:59 BAJOB  BATCON version 103(3000) running MYFILE sequence 4066 in stream 1
09:59:59 BAFIL  Input from PS:<USER=NAME>MYFILE,,1
09:59:59 BAFIL  Output to PS:<USER=NAME>MYFILE,LOG
09:59:59 BASUM  Job parameters
                Time:00:05:00  Unique:YES  Restart:NO  Output:LOG

09:59:59 MONTR
10:00:00 MONTR  SYSTEM 2102 DEVELOPMENT SYSTEM, TOPS-20 Monitor 3A(1470)
10:00:00 MONTR  $LOGIN USER=NAME 341
10:00:04 MONTR  Job 40 on TTY226 4-Apr-78 10:00:04
10:00:04 MONTR  $
10:00:04 MONTR  [CONNECTED TO PS:<USER=NAME>]
10:00:04 MONTR  $SET TIME=LIMIT 300
10:00:05 MONTR  $COMPILE MYPROG,ALG/LIST
10:00:07 USER  ALGOL: MYPROG
10:00:09 USER
10:00:09 USER  EXIT
10:00:09 MONTR  $EXECUTE MYPROG,ALG
10:00:10 USER  LINK; Loading
10:00:15 USER  [LNKXCT MYPROG Execution]
10:00:17 MONTR  1,00100006 3
10:00:17 MONTR
10:00:17 MONTR  End of execution,
10:00:17 MONTR  $C
10:00:17 MONTR  $LOGOUT
10:00:18 MONTR  Killed Job 40, User USER=NAME, Account 341, TTY 226,
10:00:18 MONTR  at 4-Apr-78 10:00:18, Used 0:0:1 in 0:0:14
10:00:21 LpDAT  [LPTLSJ LPTSPL version 103(2305) running on PLPT0, 4-Apr-78]
10:00:21 LpDAT  [LPTSJS Starting Job MYFILE, Seq #4066, request created at 4-Apr-78]
10:00:21 LpMSG  [LPTSTF Starting File PS:<SPOOL>LPT-226-0-MYPROG,LST,6]
10:00:34 LpMSG  [LPTFPF Finished Printing File PS:<SPOOL>LPT-226-0-MYPROG,LST,6]
  
```

EXAMPLES OF COMMON TASKS WITH BATCH

BASIC+2 Example

The second example is a BASIC program submitted to Batch. You can make up the program file using BASIC and save it on disk. Then make up a control file that simulates the dialogue with the BASIC system. The program is shown below.

```

@BASIC RET
READY
NEW RET
NEW PROGRAM NAME--MYBAS.BAS RET

READY
00100 INPUT D RET
00200 IF D = 2 THEN 1100 RET
00300 PRINT 'X VALUE', 'SINE', 'RESOLUTION' RET
00400 FOR X=0 TO 3 STEP D RET
00500 IF SIN(X)<=M THEN 800 RET
00600 LET X0=X RET
00700 LET M=SIN(X) RET
00800 NEXT X RET
00900 PRINT X0,M,D RET
01000 GO TO 100 RET
01100 END RET
SAVE RET

READY
MONITOR RET
@

```

The program requests data from the your terminal when it is running. You include the data in the control file. For this program, the final data item must be 2 to conclude the program. The control file follows.

```

    ESC
↓
@CREATE (FILE) BASIC.CTL RET
Input: BASIC.CTL.1
00100 @BASIC RET
00200 *OLD DSK:MYBAS.BAS RET
00300 *RUN RET
00400 *.1 RET
00500 *.01 RET
00600 *.001 RET
00700 *2 RET
00800 *MONITOR RET
00900 $
    ↑
    ESC
↓
*E RET

[BASIC.CTL.1]
@

```

EXAMPLES OF COMMON TASKS WITH BATCH

The output from the program will be printed as part of the control file listing. The command to submit the job to Batch is executed as follows.

```

      ESC
      ↓
@SUBMIT (BATCH JOB) BASIC.CTL RET
[INF:BASIC=/Seq:2972/Time:0:05:00]
@
  
```

```

08:49:29 BAJOB  BATCON version 103(3000) running BASIC sequence 2907 in stream 1
08:49:29 BAFIL  Input from PS:USER=NAME>BASIC,CTL,1
08:49:29 BAFIL  Output to PS:USER=NAME>BASIC,LOG
08:49:29 BASUM  Job parameters
                Time:00:05:00  Unique:YES  Restart:NO  Output:LOG

08:49:29 MONTR  SYSTEM 2102 DEVELOPMENT SYSTEM, TOPS=20 Monitor 3A(1475)
08:49:30 MONTR  $LOGIN USER=NAME 341
08:49:31 MONTR  Job 30 on TTY230 5-Apr-78 08:49:34
08:49:34 MONTR  $
08:49:34 MONTR  [CONNECTED TO PS:USER=NAME]
08:49:34 MONTR  $SET TIME=LIMIT 300
08:49:35 MONTR  ##BASIC
08:49:37 USER   * CAN'T TRAP CONTROL C, USE CONTROL A INSTEAD
08:49:38 USER
08:49:38 USER   READY
08:49:38 USER   OLD DSK:MYBAS,B20
08:49:42 USER
08:49:42 USER   READY
08:49:42 USER   RUN
08:49:42 USER
08:49:42 USER   MYBAS,B20
08:49:42 USER   Wednesday, April 5, 1978 08:49:42
08:49:42 USER
08:49:43 USER   * CAN'T TRAP CONTROL C, USE CONTROL A INSTEAD
08:49:44 USER   ? ,1
08:49:44 USER   ,1
08:49:44 USER   X VALUE      SINE      RESOLUTION
08:49:44 USER   1,6          0,9995736    0,1
08:49:44 USER   ? ,01
08:49:44 USER   ,01
08:49:45 USER   X VALUE      SINE      RESOLUTION
08:49:45 USER   1,57         0,9999997    0,01
08:49:45 USER   ? ,001
08:49:45 USER   ,001
08:49:46 USER   X VALUE      SINE      RESOLUTION
08:49:46 USER   1,571002      1          0,001
08:49:46 USER   ? 2
08:49:46 USER   2
08:49:46 USER
08:49:46 USER   RUNTIME: 0,676 SECS          ELAPSED TIME: 0100104
08:49:46 USER
08:49:46 USER   READY
08:49:46 USER   MONITOR
08:49:47 MONTR  0°C
08:49:47 MONTR  $LOGOUT
08:49:47 MONTR  Killed Job 30, User USER=NAME, Account 341, TTY 230,
08:49:53 MONTR  at 5-Apr-78 08:49:53, Used 01011 in 010119
08:56:13 LPDAT  [LPT1SJ LPTSPL version 103(2305) running on PLPT1, 5-Apr-78]
08:56:13 LPDAT  [LPTSJS Starting Job BASIC, Seq #2907, request created at 5-Apr-78]
  
```

EXAMPLES OF COMMON TASKS WITH BATCH

FORTRAN Example

The third example is a FORTRAN program that prints output on the line printer. You want to tell Batch in the control file to delete your relocatable binary file if an error occurs when your program is executed. If an error does not occur, you want Batch to save your relocatable binary file as it normally would. The program is shown below.

```

      ESC
      ↓
@CREATE (FILE) MYPROG.FOR RET
Input: MYPROG.FOR.1
00100  C      THIS PROGRAM CALCULATES PRIME NUMBERS. RET
00200      DO 10 I =11,50,2 RET
00300      J=1 RET
00400  4      J=J+2 RET
00500      A=J RET
00600      A=I/A RET
00700      L=I/J RET
00800      B=A-L RET
00900      IF (B) 5,10,5 RET
01000  5      IF (J.LT.SQRT(FLOAT(I))) GO TO 4 RET
01100      PRINT 105,I RET
01200  10     CONTINUE RET
01300  105    FORMAT (I4, 'IS PRIME.') RET
01400      END RET
01500      $
      ↑
      ESC
      ↓
*E RET

[MYPROG.FOR.1]
@

```

EXAMPLES OF COMMON TASKS WITH BATCH

You create the control file to compile and execute this program, deleting the relocatable binary file if there is an execution error as follows.

```

      ESC
      ↓
@CREATE (FILE) MYFOR.CTL RET
Input: MYFOR.CTL.1
00100  @COMPILE MYPROG.FOR RET
00200  @EXECUTE MYPROG.REL RET
00300  @IF(ERROR) @DELETE MYPROG.FOR RET
00400  END:!!END OF JOB RET
00500  $
      ↑
      ESC
      ↓
*E RET

[MYFOR.CTL.1]
@
```

You submit this job for execution as follows.

```

      ESC
      ↓
@SUBMIT (BATCH JOB) MYFOR.CTL RET
[INP:MYFOR=/Seq:2978/Time:0:05:00]
@
```

The program output is as follows.

```

11 IS PRIME,
13 IS PRIME,
17 IS PRIME,
19 IS PRIME,
23 IS PRIME,
29 IS PRIME,
31 IS PRIME,
37 IS PRIME,
41 IS PRIME,
43 IS PRIME,
47 IS PRIME,
```

EXAMPLES OF COMMON TASKS WITH BATCH

The log file produced by the job is as follows.

```

15:41:08 BAJOB  BATCON version 103(3000) running MYFOR sequence 4422 in stream 2
15:41:08 BAFIL  Input from PS:<USER-NAME>MYFOR,CTL,2
15:41:08 BAFIL  OUTPUT TO PS:<USER-NAME>MYFOR,LOG
15:41:08 BASUM  Job parameters
                  Time:00:05:00  Unique:YES  Restart:NO  Output:LOG

15:41:08 MONTR  SYSTEM 2102 DEVELOPMENT SYSTEM, TOPS-20 Monitor 3A(1475)
15:41:09 MONTR  @LOGIN USER=NAME 341
15:41:13 MONTR  Job 61 on TTY226 5-Apr-78 15:41:13
15:41:13 MONTR  *
15:41:13 MONTR  [CONNECTED TO PS:<USER-NAME>]
15:41:13 MONTR  @SET TIME=LIMIT 300
15:41:15 MONTR  @COMPILE MYPROG,FOR
15:41:22 USER  FORTRAN: MYPROG
15:41:28 USER  MAIN,
15:41:30 MONTR  @EXECUTE MYPROG,REL
15:41:31 USER  LINK; Loading
15:41:33 USER  [LNKXCT MYPROG Execution]
15:41:35 USER
15:41:35 USER  END OF EXECUTION
15:41:35 USER  CPU TIME: 0.11 ELAPSED TIME: 0.90
15:41:35 USER  EXIT
15:41:35 MONTR  *
15:41:35 TRUE  @IF (NOERROR)
15:41:35 BATCH  @GOTO END
15:41:35 BLABL  END;
                  |END OF JOB
15:41:35 MONTR  *C
15:41:35 MONTR  @LOGOUT
15:41:37 MONTR  Killed Job 61, User USER=NAME, Account 341, TTY 226,
15:41:37 MONTR  at 5-Apr-78 15:41:37, Used 010:2 in 010:23
15:57:37 LPDAT  [LPTLSJ LPTSPL version 103(2305) running on PLPT0, 5-Apr-78]
15:57:37 LPDAT  [LPTSJS Starting Job MYFOR, Seq #4420, request created at 5-Apr-78]
15:57:37 LPMSC  [LPTSIF Starting File PS:<SPOOL>LPT=164=0=FORLPT,DAT,3]
15:57:51 LPMSC  [LPTPF  Finished printing File PS:<SPOOL>LPT=164=0=FORLPT,DAT,3]

```

EXAMPLES OF COMMON TASKS WITH BATCH

COBOL Example

The fourth example is a COBOL program that writes record output on a magnetic tape. To have a tape drive assigned and your magnetic tape mounted on it, you must make a request to the operator. Since you do not know which drive will be assigned to your job, you must assign it in your job with a logical device name. The TMOUNT command assigns the drive to your job and associates the logical name that you specify with the physical drive assigned. The TMOUNT command also informs the operator of the name or ID number that identifies the particular tape you want mounted. (Your tape should be given to the operator or stored at the central site before you submit your job.) You create the program as follows.

```

      ESC
      ↓
@CREATE (FILE) MYPROG.CBL RET
Input: MYPROG.CBL.1
00100 IDENTIFICATION DIVISION. RET
00200 ENVIRONMENT DIVISION. RET
00300 INPUT-OUTPUT SECTION. RET
00400 FILE-CONTROL. RET
00500     SELECT OUTFIL ASSIGN TPDRIV. RET
00600 DATA DIVISION. RET
00700 FILE SECTION. RET
00800 FD OUTFIL LABEL RECORDS ARE STANDARD RET
00900     VALUE OF IDENTIFICATION IS "INFIL DAT" RET
01000     DATA RECORD IS OUTREC RET
01100     BLOCK CONTAINS 20 RECORDS. RET
01200 01 OUTREC PIC X(80). RET
01300 WORKING-STORAGE SECTION. RET
01400 77 A PIC 9999 USAGE IS COMP. RET
01500 PROCEDURE DIVISION. RET
01600 START. RET
01700     OPEN OUTPUT OUTFIL. RET
01800     MOVE ZEROS TO OUTREC. RET
01900     MOVE 1000 TO A. RET
02000 LOOP. RET
02100     WRITE OUTREC. RET
02200     SUBTRACT 1 FROM A. RET
02300     IF A IS GREATER THAN ZERO GO TO LOOP. RET
02400     CLOSE OUTFIL. RET
02500     STOP RUN. RET
02600
      $
      ↑
      ESC
      *E RET

[MYPROG.CBL.1]
@

```

EXAMPLES OF COMMON TASKS WITH BATCH

You create the control file (COBJOB) used to run the program (PROG1.CBL) as follows.

```

      ESC
      ↓
@CREATE (FILE) COBJOB RET
Input: COBJOB..1
00100  @TMOUNT TPDRIV:MAG1, RET
00200  @WRITE-ENABLED RET
00300  @ RET
00400  @COMPILE MYPROG.CBL/LIST RET
00500  @EXECUTE MYPROG.CBL RET
00600  @UNLOAD TPDRIV: RET
00700  @DEASSIGN TPDRIV: RET
00800  $
      ↑
      ESC
*E RET

[COBJOB..1]
@
  
```

You submit the job for execution as follows.

```

      ESC
      ↓
@SUBMIT (BATCH JOB) COBJOB.. RET
[INP:COBJOB=/Seq:2981/Time:0:05:00]
@
  
```

EXAMPLES OF COMMON TASKS WITH BATCH

The log file produced by COBJOB is shown below.

```

07:29:38 BAJOB  BATCON version 103(3000) running COBJOB sequence 4317 in stream 1
07:29:38 BAFIL  Input from PS:<USER=NAME>COBJOB,,1
07:29:38 BAFIL  Output to PS:<USER=NAME>COBJOB,LOG
07:29:38 BASUM  Job parameters
                  Time:00:05:00  Unique:YES  Restart:NO  Output:LOG

07:29:38 MONTR  SYSTEM 2102 DEVELOPMENT SYSTEM, TOPS-20 Monitor 3A(1475)
07:29:38 MONTR  @LOGIN USER=NAME 141
07:29:41 MONTR  Job 42 on TTY221 7-Apr-78 07:29:41
07:29:41 MONTR  @
07:29:41 MONTR  [CONNECTED TO PS:<USER=NAME>]
07:29:41 MONTR  @SET TIME=LIMIT 300
07:29:41 MONTR  @@TMQUNT TPDRIV,MAG1,
07:29:42 MONTR  @@@WRITE=ENABLED
07:29:42 MONTR  @@@
07:29:42 MONTR  [Operator notified]
07:30:67 MONTR  [MTA]: assigned)
07:30:67 MONTR  @@COMPILE MYPROG,CBL/LIST
07:30:59 USER  COBOL: MAIN [MYPROG,CBL]
07:31:05 USER  EXIT
07:31:05 MONTR  @@EXECUTE MYPROG,CBL
07:31:06 USER  LINK: Loading
07:31:10 USER  [LNKXCT MYPROG Execution]
07:31:12 USER  EXIT
07:31:12 MONTR  @@UNLOAD TPDRIV:
07:31:12 MONTR  @DEASSIGN TPDRIV:
07:31:13 MONTR  @^C
07:31:13 MONTR  @LOGOUT
07:31:14 MONTR  Killed Job 42, User USER=NAME, Account 341, TTY 221,
07:31:14 MONTR  at 7-Apr-78 07:31:14, Used 010:4 in 01:33
07:31:15 LPDAT  [LPTLSJ LPTSPL version 103(2305) running on PLPT0, 7-Apr-78]
07:31:15 LPDAT  [LPTSJS Starting Job COBJOB, Seq #4317, request created at 7-Apr-78]
07:31:16 LPMSG  [LPTSJTF Starting File PS:<SPOOL>LPT-5-0-MYPROG,LST,23]
07:31:30 LPMSG  [LPTPPF Finished Printing File PS:<SPOOL>LPT-5-0-MYPROG,LST,23]

```

EXAMPLES OF COMMON TASKS WITH BATCH

5.2 USING CARDS TO ENTER JOBS

ALGOL Example

The first job is a simple ALGOL program that writes its output into the log file because it has statements that would cause it normally to write to the terminal. The program is as follows.

```
BEGIN
  REAL X; INTEGER I;
  X :=1;
  FOR I :=1 UNTIL 1000 DO X :=X+I;
  PRINT (X);
END
```

The cards to enter this program are shown in Figure 5-1.

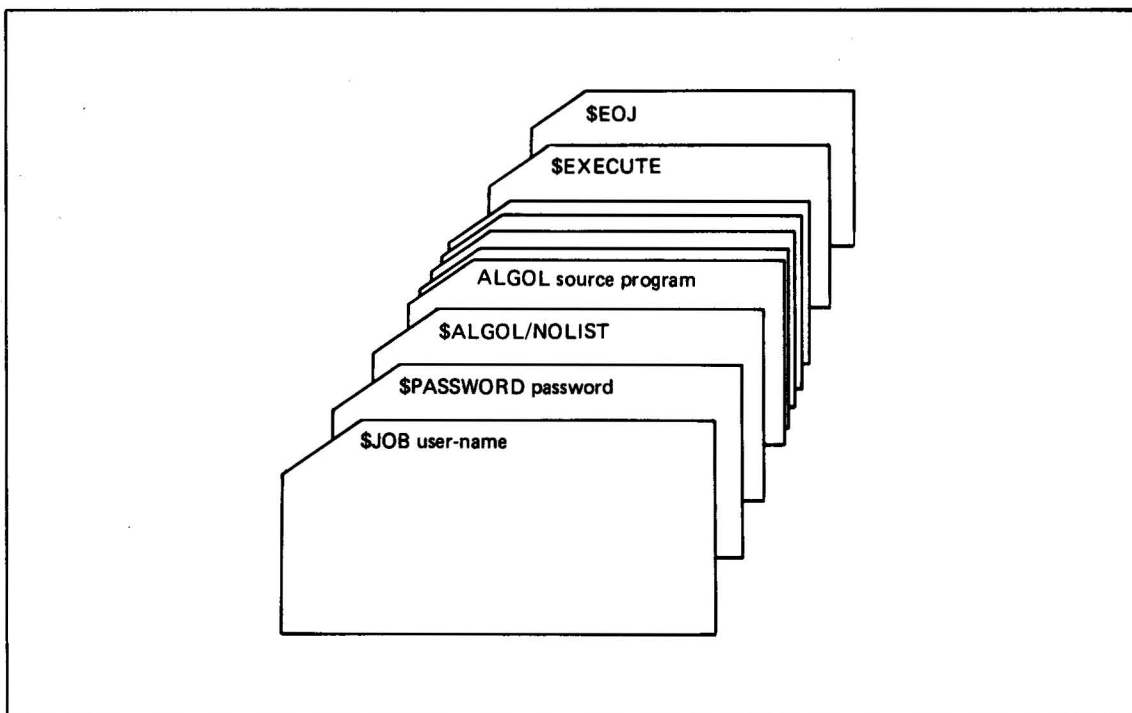


Figure 5-1 ALGOL Job Entry Card Deck

The output, including the log file, is shown on the following page.

EXAMPLES OF COMMON TASKS WITH BATCH

```

07:02:27 BAJOB  BATCON Version 103(3000) running JB2REB sequence 4302 in stream 1
07:02:27 BAFIL  Input from PS: <USER=NAME>JB2REB,CTL,1
07:02:27 BAFIL  Output to PS: <USER=NAME>JB2REB,LOG,1
07:02:27 BATSUM  Job parameters
                    Time:00:05:00  Unique:YES  Restart:YES  Output:LOG

07:02:27 MONTR  SYSTEM 2102 DEVELOPMENT SYSTEM, TOPS=20 Monitor 3A(1475)
07:02:28 MONTR  @LOGIN USER=NAME 341
07:02:31 MONTR  Job 38 on TTY22: 7-Apr-78 07:02:31
07:02:31 MONTR  @
07:02:31 MONTR  [CONNECTED TO PS: <USER=NAME>]
07:02:31 MONTR  @SET TIME=LIMIT 300
07:02:31 MONTR  @#COMPILE /COMP/ALG PS:LN2REA,ALG
07:02:32 USER   ALGOL: LN2REA
07:02:33 USER
07:02:33 USER   EXIT
07:02:33 MONTR  @#EXECUTE /REL PS:LN2REA,REL
07:02:34 USER   LINK: Loading
07:02:36 USER   [LNKXCT LN2REA Execution]
07:02:37 MONTR  1,00100004 3
07:02:37 MONTR
07:02:37 MONTR  End of execution,
07:02:37 MONTR  @
                    %ERR:1
                    %FIN:1
07:02:37 BLABL  @DELETE PS:LN2REA,ALG,PS:LN2REA,REL
07:02:37 MONTR  <USER=NAME>LN2REA,ALG,1 [OK]
07:02:37 MONTR  <USER=NAME>LN2REA,REL,1 [OK]
07:02:37 MONTR  @*C
07:02:37 MONTR  @LOGOUT
07:02:38 MONTR  Killed Job 38, User USER=NAME, Account 341, TTY 221,
07:02:38 MONTR  at 7-Apr-78 07:02:38, Used 0:0:1 in 0:0:7
07:02:39 LPDAT  [LPTLSJ LPTSPL version 103(2305) running on PLPT1, 7-APR-78]
07:02:39 LPDAT  [LPTSJS Starting Job JB2REB, Seq #4302, request created at 7-Apr-78]

```

EXAMPLES OF COMMON TASKS WITH BATCH

BASIC+2 Example

The next example shows how to enter a BASIC program. You must precede the program commands with a \$CREATE card so that the program will be copied into a file on disk. No \$DATA card can be used because BASIC does not use the EXECUTE command and because the data is entered by means of the control file: the program requests data when it is running; it finds the data in the control file. For this program the final data item in the control file must be 2 so that the program can be concluded. The program is shown below.

```
5    INPUT D
10   IF D=2 THEN 100
20   PRINT "X VALUE", "SINE", "RESOLUTION"
30   FOR X=0 TO 3 STEP D
40   IF SIN(X)=M THEN 80
50   LET X0=X
60   LET M=SIN(X)
70   NEXT X
80   PRINT X0,M,D
90   GO TO 5
100  END
```

The cards to enter the program and run it are shown in Figure 5-2.

EXAMPLES OF COMMON TASKS WITH BATCH

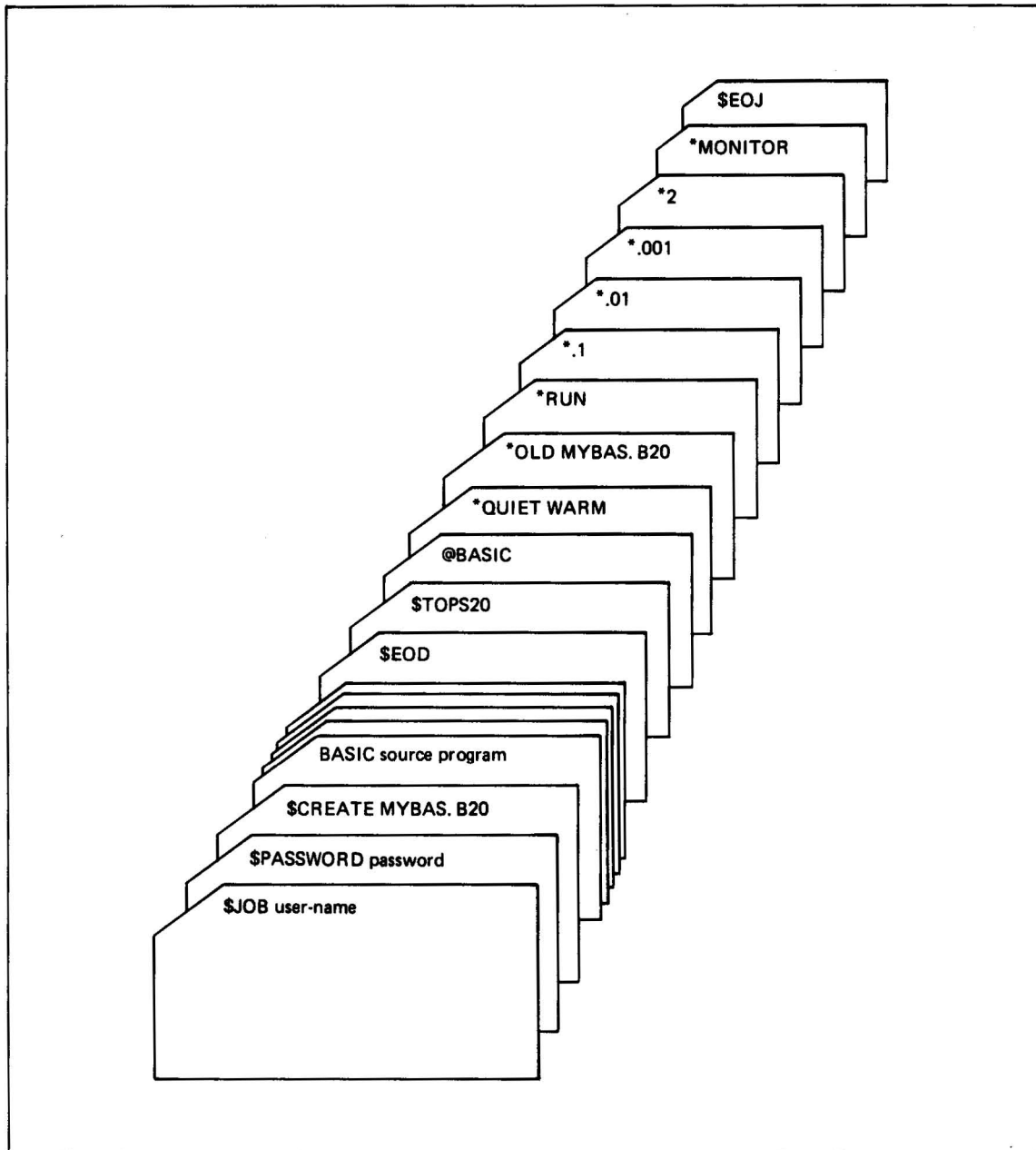


Figure 5-2 BASIC Job Entry and Run Card Deck

EXAMPLES OF COMMON TASKS WITH BATCH

The output from the program will be printed in the log file because it would normally be printed on the terminal. The log file is shown below.

```
10:05:52 STDAT 7-APR-78 SYSTEM 2102 DEVELOPMENT SYSTEM, TOPS-20 Monitor 3A(1475)
10:05:52 STCRD $JOB USER=NAME /ACCOUNT:341
10:05:52 STCRD $CREATE MYBAS,B20
10:05:54 STMSG File PS:MYBAS,B20 Created - 11 Cards Read
10:05:54 STCRD $TOPS20
10:05:55 STCRD $EOJ
10:05:55 STSUM End of Job Encountered
10:05:55 STSUM 25 Cards Read
10:05:55 STSUM Batch Input Request Created
```

```
10:15:40 BAJOB BATCON version 103(3000) running JB2RDW sequence 4355 in stream 1
10:15:40 BAFIL Input from PS:USER=NAME>JB2RDW,CTL,1
10:15:40 BAFIL Output to PS:USER=NAME>JB2RDW,LOG,1
10:15:40 BASUM Job parameters
Time:00:05:00 Unique:YES Restart:YES Output:LOG
```

```
10:15:40 MONTR
10:15:40 MONTR SYSTEM 2102 DEVELOPMENT SYSTEM, TOPS-20 Monitor 3A(1475)
10:15:41 MONTR $LOGIN USER=NAME 341
10:15:44 MONTR Job 34 on TTY230 7-APR-78 10:15:44
10:15:44 MONTR
10:15:44 MONTR [CONNECTED TO PS:USER=NAME>]
10:15:44 MONTR $SET TIME=LIMIT 300
10:15:44 MONTR ***BASIC
10:15:45 USER * CAN'T TRAP CONTROL C, USE CONTROL A INSTEAD
10:15:45 USER * CAN'T TRAP CONTROL C, USE CONTROL A INSTEAD
10:15:46 USER
10:15:46 USER READY
10:15:46 USER *QUIET WARN
10:15:46 USER
10:15:46 USER READY
10:15:46 USER *OLD MYBAS,B20
10:15:46 USER
10:15:47 USER READY
10:15:47 USER *RUN
10:15:48 USER
10:15:48 USER MYBAS,B20
10:15:48 USER Friday, April 7, 1978 10:15:47
10:15:48 USER
10:15:48 USER * CAN'T TRAP CONTROL C, USE CONTROL A INSTEAD
10:15:48 USER ?
10:15:49 USER *,1
10:15:49 USER X VALUE SINE RESOLUTION
10:15:49 USER 0 0 0.1
10:15:49 USER ?
10:15:49 USER *,01
10:15:49 USER X VALUE SINE RESOLUTION
10:15:49 USER 0 0 0.01
10:15:49 USER ?
10:15:49 USER *,001
10:15:49 USER X VALUE SINE RESOLUTION
10:15:49 USER 0 0 0.001
10:15:49 USER ?
10:15:49 USER *2
10:15:49 USER
10:15:49 USER RUNTIME: 0.444 SECS ELAPSED TIME: 0:00:01
10:15:49 USER
10:15:49 USER READY
10:15:49 USER
10:15:49 USER *MONITOR
10:15:49 USER
10:15:50 MONTR $ERR:1
```

BTNECF End of the Control File while searching for \$FIN

```
10:15:50 MONTR *C
10:15:50 MONTR $LOGOUT
10:15:55 MONTR Killed Job 34, User USER=NAME, Account 341, TTY 230,
10:15:55 MONTR at 7-APR-78 10:15:55, Used 0:0:1 in 0:0:11
10:16:00 LPDAT [LPTLSJ LPTSPL version 103(2305) running on PLPT1, 7-APR-78]
10:16:00 LPDAT [LPTSJS Starting Job JB2RDW, Seq #4355, request created at 7-APR-78]
```

EXAMPLES OF COMMON TASKS WITH BATCH

FORTRAN Example

The third example shows a FORTRAN program that prints output on the line printer. In the control file, you want to tell Batch to prevent execution if the program compiles incorrectly.

```
C   THIS PROGRAM CALCULATES PRIME NUMBERS FROM 11 TO 50.
    DO 10 I=11,50,2
      J=1
4     J=J+2
      A=J
      A=I/A
      L=I/J
      B=A-L
      IF (B) 5,10,5
5     IF (J.LT.SQRT(FLOAT(I))) GO TO 4
      PRINT 105,I
10    CONTINUE
105   FORMAT (I4, 'IS PRIME.')
      END
```

The cards used to enter this program are shown in Figure 5-3.

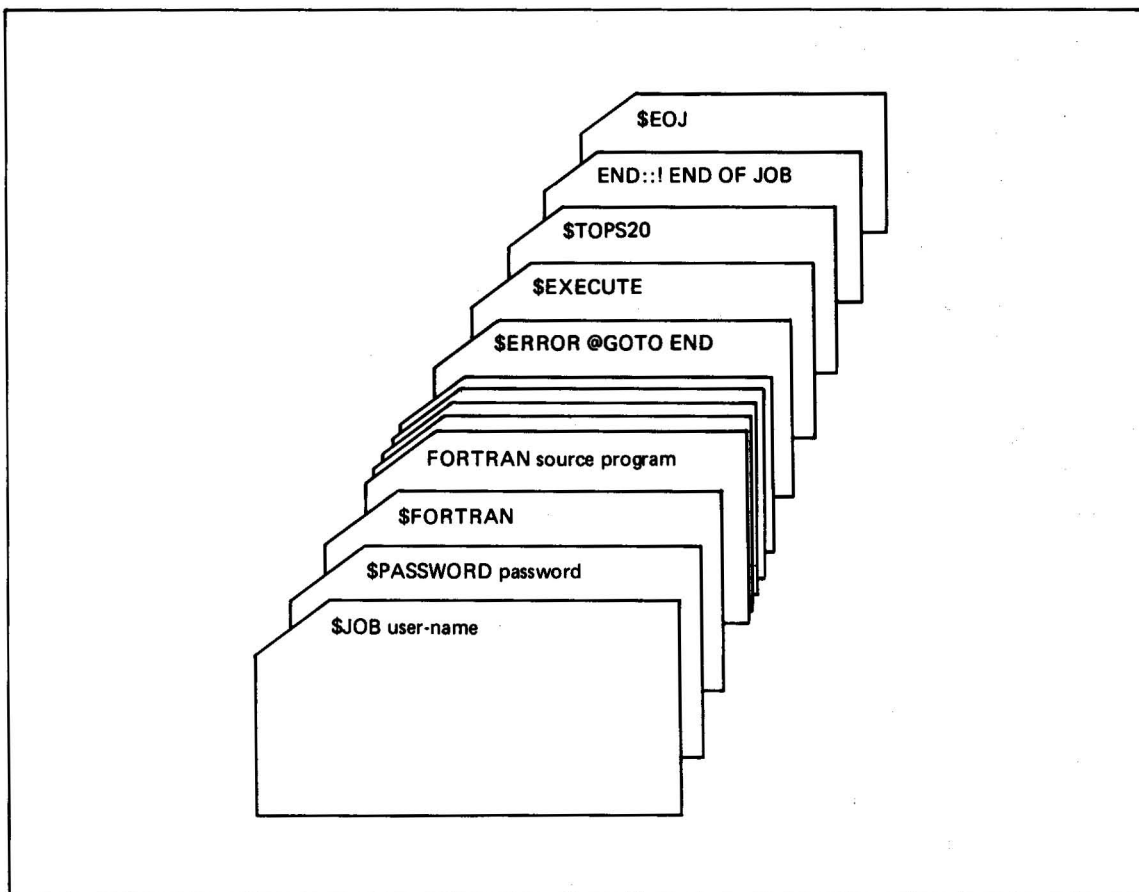


Figure 5-3 FORTRAN Card Deck That Prevents Execution on Error

EXAMPLES OF COMMON TASKS WITH BATCH

Batch puts the following commands into the control file as a result of the cards you entered.

```
@COMPILE LNxxxx.FOR/COMPILE/LIST
@IF (ERROR) @GOTO END
@EXECUTE
END::!END OF JOB
```

Program output is as follows.

```
11 IS PRIME,
13 IS PRIME,
17 IS PRIME,
19 IS PRIME,
23 IS PRIME,
29 IS PRIME,
31 IS PRIME,
37 IS PRIME,
41 IS PRIME,
43 IS PRIME,
47 IS PRIME,
```

The log file produced by the job is on the following page.

EXAMPLES OF COMMON TASKS WITH BATCH

```
07:01:53 STDAT 7=APR=78 SYSTEM DEVELOPMENT SYSTEM, TOPS=20 Monitor JA(1475)
07:01:53 STCRD $JOB USER=NAME /ACCOUNT:341
07:01:53 STCRD $FORTRAN/LIST
07:01:55 STMSG File PS:LN2RE7,FOR Created = 14 Cards Read
07:01:55 STCRD $ERROR $GOTO END
07:01:55 STCRD $EXECUTE
07:01:55 STCRD $TOPS20
07:01:55 STCRD $EOJ
07:01:55 STSUM End of Job Encountered
07:01:55 STSUM 22 Cards Read
07:01:55 STSUM Batch Input Request Created
```

```
07:03:21 BAJOB BATCON version 103(3000) running JB2RE8 sequence 4304 in stream 1
07:03:21 BAFIL Input from PS:<USER=NAME>JB2RE8,CTL,1
07:03:21 BAFIL Output to PS:<USER=NAME>JB2RE8,LOG,1
07:03:21 BASUM Job parameters
Time:00:05:00 Unique:YES Restart:YES Output:LOG
```

```
07:03:21 MONTR
07:03:21 MONTR SYSTEM 2102 DEVELOPMENT SYSTEM, TOPS=20 Monitor 3A(1475)
07:03:21 MONTR $LOGIN USER=NAME 341
07:03:24 MONTR Job 38 on TTY22: 7=Apr=78 07:03:24
07:03:24 MONTR $
07:03:24 MONTR [CONNECTED TO PS:<USER=NAME>]
07:03:24 MONTR $SET TIME=LIMIT 300
07:03:24 MONTR $*COMPILE /COMP/FOR PS:LN2RE7,FOR/LIST
07:03:26 USER FORTRAN: LN2RE7
07:03:29 USER MAIN,
07:03:30 MONTR $
07:03:30 FALSE $IF(ERROR) $GOTO END
07:03:30 MONTR $EXECUT /REL PS:LN2RE7,REL
07:03:30 MONTR EXECUT /REL PS:LN2RE7,REL
07:03:31 USER LINK: Loading
07:03:33 USER [LNKXCT LN2RE7 Execution]
07:03:34 USER
07:03:34 USER END OF EXECUTION
07:03:34 USER CPU TIME: 0.10 ELAPSED TIME: 0.88
07:03:34 USER EXIT
07:03:34 MONTR $
07:03:34 BLABL END:;
JEND OF JOB
$ERR:;
$FIN:;
07:03:34 BLABL $DELETE PS:LN2RE7,FOR,PS:LN2RE7,REL
07:03:34 MONTR <USER=NAME>LN2RE7,FOR,1 [OK]
07:03:34 MONTR <USER=NAME>LN2RE7,REL,1 [OK]
07:03:34 MONTR $*C
07:03:35 MONTR $LOGOUT
07:03:35 MONTR Killed Job 38, User USER=NAME, Account 341, TTY 221,
07:03:35 MONTR at 7=Apr=78 07:03:35, Used 010:2 in 010:10
07:03:36 LPDAT [LPTLSJ LPTSPL version 103(2305) running on PLPT0, 7=Apr=78]
07:03:36 LPDAT [LPTSJS Starting Job JB2RE8, Seq #4304, request created at 7=Apr=78]
07:03:37 LPMSG [LPTSTF Starting File PS:<SPOOL>LPT=226=0=LN2RE7,,17]
07:03:50 LPMSG [LPTFPF Finished Printing File PS:<SPOOL>LPT=226=0=LN2RE7,,17]
07:03:51 LPMSG [LPTSTF Starting File PS:<SPOOL>LPT=226=0=FORLPT,DAT,18]
07:04:02 LPMSG [LPTFPF Finished Printing File PS:<SPOOL>LPT=226=0=FORLPT,DAT,18]
```

EXAMPLES OF COMMON TASKS WITH BATCH

COBOL Example

The fourth example is a COBOL program that writes record output on a magnetic tape. To have a tape drive assigned and your magnetic tape mounted on it, you must make a request to the operator. Since you do not know which drive will be assigned to your job, you must assign it in your job with a logical device name. The TMOUNT command assigns the drive to your job and associates the logical name that you specify with the physical drive assigned. The TMOUNT command also informs the operator of the name or ID number that identifies the particular tape you want mounted. (Your tape should be given to the operator or stored at the central site before you submit your job.) The program is as follows.

```
IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT OUTFIL ASSIGN TPDRIV.
DATA DIVISION.
FILE SECTION.
FD OUTFIL LABEL RECORDS ARE STANDARD
    VALUE OF IDENTIFICATION IS "INFIL DAT"
    DATA RECORD IS OUTREC
    BLOCK CONTAINS 20 RECORDS.
01 OUTREC PIC X(80).
WORKING-STORAGE SECTION.
77 A PIC 9999 USAGE IS COMP.
PROCEDURE DIVISION.
START.
    OPEN OUTPUT OUTFIL.
    MOVE ZEROS TO OUTREC.
    MOVE 1000 TO A.
LOOP.
    WRITE OUTREC.
    SUBTRACT 1 FROM A.
    IF A IS GREATER THAN ZERO GO TO LOOP.
    CLOSE OUTFIL.
    STOP RUN.
```

The cards to enter this job are shown in Figure 5-4.

EXAMPLES OF COMMON TASKS WITH BATCH

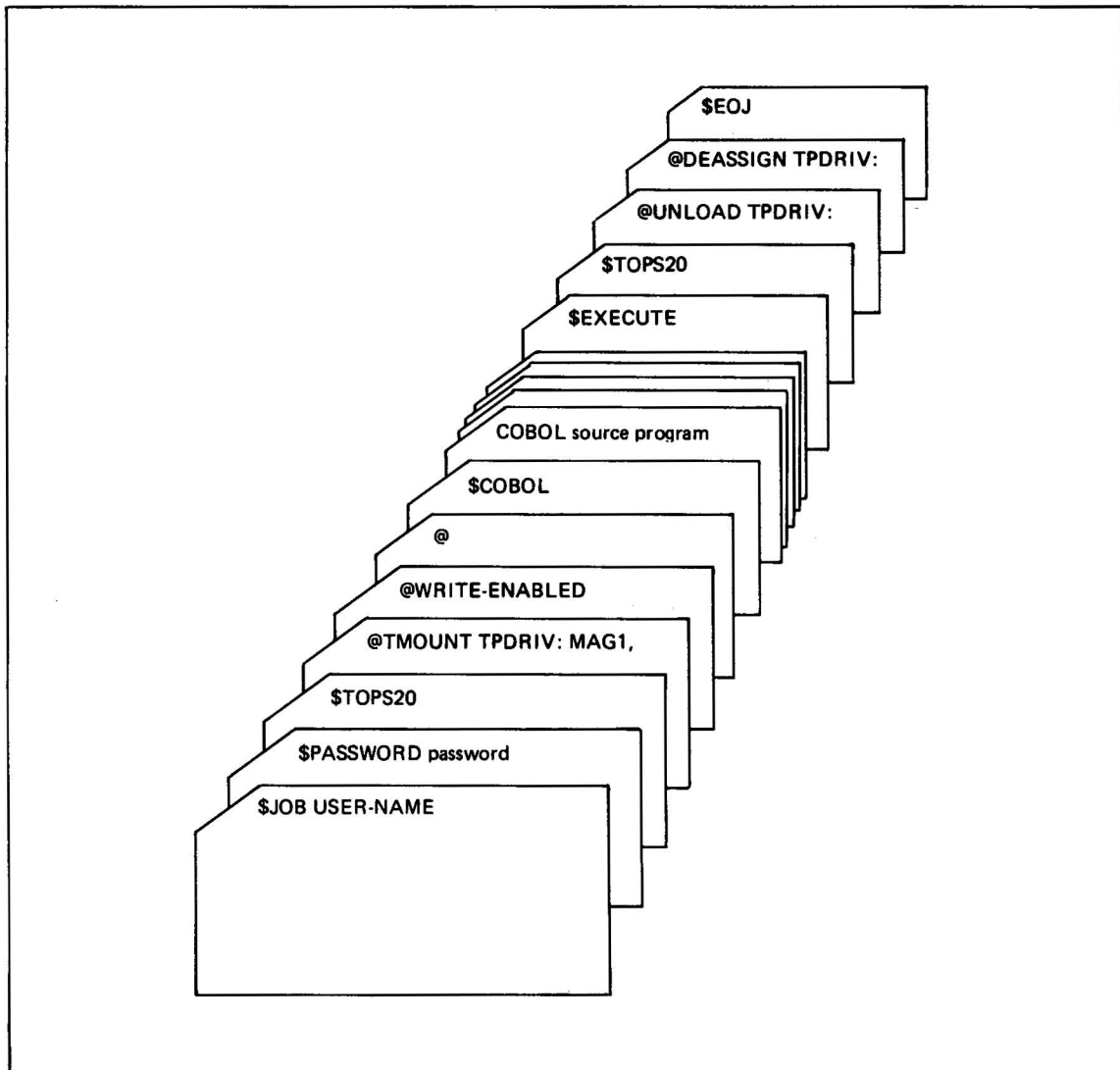


Figure 5-4 COBOL Program Card Deck Using Data From Magnetic Tape

Batch puts the following commands into the control file for you.

```
@TMOUNT TPDRIV:MAG1,
@WRITE-ENABLED
@
@COMPILE/COMP/COB DSK:LNxxxx.REL
@EXECUTE/REL DSK:LNxxxx.REL
@UNLOAD TPDRIV:
@DEASSIGN TPDRIV:
```

EXAMPLES OF COMMON TASKS WITH BATCH

```

07:24:06 BAJOB BATCON version 103(3000) running JB2RE0 sequence 4315 in stream 1
07:24:06 BAFIL Input from PS:<USER=NAME>JB2RE0,CTL,1
07:24:06 BAFIL Output to PS:<USER=NAME>JB2RE0,LOG,1
07:24:06 BASUM Job parameters
Time:00:05:00 Unique:YES Restart:YES Output:LOG

07:24:06 MONTR
07:24:06 MONTR SYSTEM 2102 DEVELOPMENT SYSTEM, TOPS=20 Monitor 3A(1475)
07:24:06 MONTR @LOGIN USER=NAME 341
07:24:09 MONTR Job 42 on TTY221 7-Apr-78 07:24:09
07:24:09 MONTR @
07:24:09 MONTR [CONNECTED TO PS:<USER=NAME>]
07:24:09 MONTR @SET TIME=LIMIT 300
07:24:10 MONTR @@TMOUNT TPDRIV,MAG1,
07:24:10 MONTR @@@WRITE=ENABLED
07:24:10 MONTR @@@
07:24:10 MONTR [Operator notified]
07:24:10 MONTR [MTA3: assigned]
07:24:33 MONTR @@COMPIL /COMP/COB PS:LN2RDZ,CBL/LIST
07:24:34 USER CUSOL: MAIN [LN2RDZ,CBL]
07:24:40 USER
07:24:40 USER EXIT
07:24:40 MONTR @@@EXECUT /REL PS:LN2RDZ,REL
07:24:41 USER LINK: Loading
07:24:45 USER [LNKXCT LN2RDZ Execution]
07:24:48 USER
07:24:48 USER EXIT
07:24:48 MONTR @@@UNLOAD TPDRIV:
07:24:48 MONTR @@@DEASSIGN TPDRIV:
07:24:48 MONTR @
07:24:48 MONTR $ERR:1
07:24:48 MONTR $FIN:1
07:24:48 MONTR @DELETE PS:LN2RDZ,CBL,PS:LN2RDZ,REL
07:24:48 MONTR <USER=NAME>LN2RDZ,CBL,1 [OK]
07:24:48 MONTR <USER=NAME>LN2RDZ,REL,1 [OK]
07:24:48 MONTR @*C
07:24:49 MONTR @LOGOUT
07:24:50 MONTR Killed Job 42, User USER=NAME, Account 341, TTY 221,
07:24:50 MONTR at 7-Apr-78 07:24:50, Used 0:0:4 in 0:0:40
07:24:51 LPDAT [LPTLSJ LPTSPL version 103(2305) running on PLPT0, 7-Apr-78]
07:24:51 LPDAT [LPTLSJ Starting Job JB2RE0, Seq #4315, request created at 7-Apr-78]
07:24:51 LPMSC [LPTSTF Starting File PS:<SPOOL>LPT-226-0=LN2RDZ,LST,22]
07:25:07 LPMSC [LPTPPF Finished Printing File PS:<SPOOL>LPT-226-0=LN2RDZ,LST,22]

```


INDEX

- ! character, 2-3, 3-2
- \$ character, 3-2
- * character, 2-3, 3-2
- character, 3-2, 3-4
- : character, 3-4
- = character, 2-3
- @ character, 2-3, 3-2

- \$-language card, 3-1, 3-7, 3-8
- \$-language card examples, 3-9
- \$-language card /switches, 3-8
- /AFTER switch, 2-5, 3-3, 3-5
- ALGOL, Glossary-1
- \$ALGOL, 3-7, 3-8
- ALGOL batch log file, 5-2, 5-12
- ALGOL card-input example, 5-11, 5-12
- ALGOL example, 3-14, 3-15
- ALGOL terminal example, 5-1, 5-2
- Alphanumeric, Glossary-1
- ASCII Code, Glossary-1
- Assemble, Glossary-1
- Assembly Language, Glossary-1
- Assembly Listing, Glossary-1

- @BACKTO command, 2-11
- @BACKTO label, 2-11
- BASIC, Glossary-1
- BASIC batch process, 3-23
- BASIC program, 3-23
- BASIC program in batch, 3-24, 3-25
- BASIC+2 batch log file, 5-4, 5-15,
- BASIC+2 card-input example, 5-13, 5-14, 5-15
- BASIC+2 terminal example, 5-3, 5-4
- Batch,
 - BASIC program in, 3-24, 3-25
 - common tasks with, 5-1
 - how to use, 1-2
 - summary of, 1-3
 - using the terminal for, 5-1
 - what is, 1-1
- Batch card commands, 3-1
- Batch commands, 1-2, 2-8, 3-1
- Batch control card commands, 3-2
- Batch control card format, 3-2
- Batch job,
 - card, 3-1
 - entering a, 2-1, 3-1
 - submitting, 1-3, 2-4
 - terminal, 5-1
- Batch output, 4-1
 - additional, 4-2
 - sample, 4-2, 4-3
- Batch printed output, 4-1
- Batch procedure,
 - terminal, 2-2
- Batch process,
 - BASIC, 3-23
- Batch Processing, Glossary-1
- Batch system,
 - TOPS-20, 1-1

- Card, Glossary-2
- Card,
 - \$-language, 3-1, 3-7, 3-8
 - \$CREATE, 3-1, 3-6
 - Column, Glossary-2
 - \$DATA, 3-1, 3-10, 3-14
 - \$EOD, 3-1, 3-16
 - \$EOJ, 3-1, 3-6
 - \$ERROR, 3-1, 3-18
 - \$EXECUTE, 3-1, 3-10
 - Field, Glossary-2
 - \$JOB, 3-1, 3-2
 - \$NOERROR, 3-1, 3-18
 - \$PASSWORD, 3-1, 3-5
 - Row, Glossary-2
 - \$TOPS20, 3-1, 3-16
- Card batch job, 3-1
- Card commands,
 - batch, 3-1
- Card comments, 3-2
- Card deck,
 - commands in the, 3-21
 - print program, 4-5
 - program, 3-20
 - setting up your, 3-20
- Card error recovery
 - procedure, 3-29, 3-30, 3-31
- Card error recovery program, 3-28

INDEX (CONT.)

- Card file,
 - error recovery in, 3-26, 3-27
- Card format conventions, 3-2
- Card input, 1-1, 5-11
 - output from, 4-5, 4-6, 4-7
- Card /switches, \$JOB, 3-3
- Card user-name, \$JOB, 3-3
- Card-reader file, spooled, 3-12
- Cards to enter jobs, using, 5-11
- Central Processing Unit, Glossary-2
- Character, Glossary-2
- Character, @ERROR, 2-9
- Characters, special, 2-3
- COBOL, Glossary-2
- \$COBOL, 3-7, 3-8
- COBOL batch log file, 5-10, 5-21
- COBOL card-input example, 5-19, 5-20
- COBOL example, 3-13, 3-14
- COBOL terminal example, 5-8, 5-9, 5-10
- Command, Glossary-2
- Command,
 - @BACKTO, 2-11
 - @ERROR, 2-9
 - @GOTO, 2-10
 - @IF, 2-8
 - @NOERROR, 2-9
 - @SUBMIT, 2-5, 2-7
- Commands,
 - batch, 1-2, 2-8
 - batch control card, 3-2
 - invalid, 3-22
 - system, 1-2, 3-16
 - system program, 1-2
 - TOPS-20, 2-4
 - TOPS-20 EXEC, 2-2
 - using TOPS-20, 3-18
- Commands in the card deck, 3-21
- Comment card, 3-2
- Common tasks with batch, 5-1
- Comparing two card decks, 3-22
- Compile, Glossary-2
- Compiler, Glossary-2
- Compiling a program, 3-7
- Computer, Glossary-2
- Computer Operator, Glossary-2
- Continuation Card, Glossary-2
- Control card commands, batch, 3-2
- Control card file, batch, 3-21
- Control card format, batch, 3-2
- Control cards,
 - nonspecial, 3-22
 - special, 3-22
- Control File, Glossary-3
- Control file, 1-2
 - creating, 1-3, 2-1, 2-2
 - error recovery in, 2-12, 2-13
- Control file format, 2-3
- Control-file-specification, 2-5
- Conventions,
 - card format, 3-2
- CPU, Glossary-3
- \$CREATE card, 3-1, 3-6
- \$CREATE card examples, 3-7
- \$CREATE card filename.typ, 3-6
- \$CREATE card /switches, 3-6
- Creating the control file, 1-3, 2-1, 2-2, 3-6
- Cross-Reference Listing, Glossary-3
- Data, Glossary-3
- Data,
 - executing a program with, 3-10
 - input, 2-2
- \$DATA card, 3-1, 3-10, 3-14
- \$DATA card examples, 3-12
- \$DATA card filename.typ, 3-11
- \$DATA card format, 3-11
- \$DATA card /switches, 3-11
- Data files,
 - naming, 3-14
- Data input,
 - end of, 3-16
- Debug, Glossary-3
- Disk, Glossary-3
- EDIT,
 - using, 2-2
- End of data input, 3-16
- Ending a job, 3-6

INDEX (CONT.)

Enter jobs,
 using cards to, 5-11
 using terminal to, 5-1
 Entering a batch job, 2-1,
 3-1
 \$EOD card, 3-1, 3-16
 \$EOJ card, 3-1, 3-6
 \$ERROR card, 3-1, 3-18
 \$ERROR card format, 3-19
 \$ERROR card statement, 3-19
 @ERROR character, 2-9
 @ERROR command, 2-9
 Error recovery, 2-3, 3-18
 Error recovery,
 specifying, 2-12, 2-13,
 3-26, 3-27
 Error recovery in card file,
 3-26, 3-27
 Error recovery in control
 file, 2-12, 2-13
 Error recovery procedure,
 card, 3-29, 3-30, 3-31
 Error recovery program,
 card, 3-28
 Errors,
 recovering from, 1-3
 Examples, 5-1
 \$-language card, 3-9
 ALGOL, 3-14, 3-15
 ALGOL card-input, 5-11,
 5-12
 ALGOL terminal, 5-1, 5-2
 BASIC+2 card-input, 5-13,
 5-14, 5-15
 BASIC+2 terminal, 5-3,
 5-4
 COBOL, 3-13, 3-14
 COBOL card-input, 5-19,
 5-20
 COBOL terminal, 5-8, 5-9,
 5-10
 \$CREATE card, 3-7
 \$DATA card, 3-12
 FORTRAN, 3-13, 3-15
 FORTRAN card-input, 5-17,
 5-18
 FORTRAN terminal, 5-5,
 5-7
 log file, 4-7
 submitting job, 2-6, 2-7
 EXEC commands,
 TOPS-20, 2-2
 Execute, Glossary-3
 \$EXECUTE card, 3-1, 3-10
 \$EXECUTE card /switches,
 3-10
 Executing a program, 3-10
 Executing a program with
 data, 3-10

FILCOM program,
 TOPS-20, 2-4
 File, Glossary-3
 File,
 control, 1-2
 control card, 3-21
 creating a, 3-6
 log, 1-2, 2-2, 4-1
 Filename, Glossary-3
 Filename.typ,
 \$CREATE card, 3-6
 \$DATA card, 3-11
 File Type, Glossary-3
 Format,
 \$-language card, 3-8
 batch control card, 3-2
 control file, 2-3
 \$CREATE card, 3-6
 \$DATA card, 3-11
 \$EOD card, 3-16
 \$EOJ card, 3-6
 \$ERROR card, 3-19
 \$EXECUTE card, 3-10
 \$JOB card, 3-3
 \$NOERROR card, 3-19
 \$PASSWORD card, 3-5
 \$TOPS20 card, 3-17
 Format conventions,
 card, 3-2
 Format of lines, 2-3
 FORTRAN, Glossary-3
 \$FORTRAN, 3-7, 3-8
 FORTRAN batch log file, 5-7
 5-18
 FORTRAN card-input example,
 5-16, 5-17
 FORTRAN card-input output,
 5-17
 FORTRAN example, 3-13, 3-15
 FORTRAN terminal example,
 5-5, 5-7

Generation Number,
 Glossary-3
 @GOTO command, 2-10
 @GOTO label, 2-10

How to use batch, 1-2

Identifying yourself, 3-5
 @IF command, 2-8
 @IF (ERROR) statement, 2-8

INDEX (CONT.)

- @IF (NOERROR) statement, 2-8
- Input data, 2-2
- Interpreting your printed output, 4-1
- Introduction, 1-1
- Invalid commands, 3-22

- Job, Glossary-3
- Job,
 - output from your, 4-1
 - running your, 1-2
- \$JOB card, 3-1, 3-2
- \$JOB card /switches, 3-3
- \$JOB card user-name, 3-3
- /JOBNAME switch, 3-3

- Label, Glossary-4
- Label,
 - @BACKTO, 2-11
 - @GOTO, 2-10
- Line-printer output, 4-1
- Lines,
 - format of, 2-3
- Log File, Glossary-4, 1-2, 2-2, 4-1
- Log file,
 - ALGOL batch, 5-2, 5-12
 - BASIC+2 batch, 5-4, 5-15, 5-16
 - COBOL batch, 5-10, 5-21
 - FORTRAN batch, 5-7, 5-18
- Log file example, 4-7

- MACRO, Glossary-4
- \$MACRO, 3-7, 3-8
- /MAP switch, 3-10, 3-12
- Message,
 - page limit exceeded, 2-5
 - ?TIME-LIMIT-EXCEEDED, 2-10
- Mounting a Device, Glossary-4

- Naming data files, 3-14
- \$NOERROR card, 3-1, 3-18
- \$NOERROR card format, 3-19
- \$NOERROR card statement, 3-19
- @NOERROR command, 2-9

- /NOLIST switch, 3-9
- Nonspecial control cards, 3-22

- Object Program, Glossary-4
- Output,
 - batch, 4-1
 - FORTRAN card-input, 5-17
 - line-printer, 4-1
 - printed, 4-2, 4-3
 - receiving your, 1-3
- Output from card input, 4-5, 4-6, 4-7
- Output from terminal input, 4-2, 4-3, 4-4
- Output from your job, 4-1

- Page limit exceeded message, 2-5
- /PAGE switch, 2-5, 3-3, 3-4
- Password, Glossary-4, 3-5
- \$PASSWORD card, 3-1, 3-5
- Peripheral Device, Glossary-4
- Print program card deck, 4-5
- /PRINT switch, 3-7
- Printed output, 4-2, 4-3
- Printed output,
 - batch, 4-1
 - interpreting your, 4-1
- Procedure,
 - card error recovery, 3-29, 3-30, 3-31
 - terminal batch, 2-2
- Process,
 - BASIC batch, 3-23
- Program, Glossary-4
- Program,
 - BASIC, 3-23
 - card error recovery, 3-28
 - compiling a, 3-7
 - executing a, 3-10
- Program card deck, 3-20
- Program commands,
 - system, 3-1
- Program in batch,
 - BASIC, 3-24, 3-25
- Programming, Glossary-4
- Program with data,
 - executing a, 3-10

- Queue, Glossary-4, 1-2

INDEX (CONT.)

Reading from spooled
 card-reader, 3-12
 Receiving your output, 1-3
 Recovering from errors, 1-3
 Recovery,
 error, 2-3, 3-18
 Running your job, 1-2

 Sample batch output, 4-2,
 4-3
 Setting up your card deck,
 3-20
 Software, Glossary-4
 Source Deck, Glossary-4
 Source Language, Glossary-4
 Source Program, Glossary-5
 Special characters, 2-3
 Special control cards, 3-22
 Specifying error recovery,
 2-12, 2-13, 3-26, 3-27
 Spooled card-reader,
 reading from, 3-12
 Spooled card-reader file,
 3-12
 Starting a job, 3-2
 Statement,
 \$ERROR card, 3-19
 @IF (ERROR), 2-8
 @IF (NOERROR), 2-8
 \$NOERROR card, 3-19
 Submit batch job, 1-3
 SUBMIT command, 2-5, 2-7
 SUBMIT /switches, 2-5
 Submitting job examples,
 2-6, 2-7
 Submitting the batch job,
 2-4
 Summary of batch, 1-3
 /SUPPRESS switch, 3-7, 3-9,
 3-11, 3-17
 Switches,
 \$-language card, 3-8
 /AFTER, 2-5, 3-3, 3-5
 \$CREATE card, 3-6
 \$DATA card, 3-11
 \$EXECUTE card, 3-10
 \$JOB card, 3-3
 /JOBNAME, 3-3
 /MAP, 3-10, 3-12
 /NOLIST, 3-9
 /PAGE, 2-5, 3-3, 3-4
 /PRINT, 3-7
 @SUBMIT, 2-5
 /SUPPRESS, 3-7, 3-9, 3-11,
 3-17
 /TIME, 2-6, 3-4
 \$TOPS20 card, 3-17

Switches (Cont.),
 /WIDTH, 3-7, 3-8, 3-11,
 3-17
 System, Glossary-5
 System Command, Glossary-5
 System commands, 1-2, 3-16
 TOPS-20, 3-1
 System Program, Glossary-5
 System program commands,
 1-2, 3-1

Terminal, Glossary-5
 Terminal,
 timesharing, 2-1
 Terminal batch jobs, 5-1
 Terminal batch procedure,
 2-2
 Terminal for batch,
 using the, 5-1
 Terminal input, 1-1, 2-1
 Terminal input,
 output from, 4-2, 4-3,
 4-4
 /TIME switch, 2-6, 3-4
 ?TIME-LIMIT-EXCEEDED
 message, 2-10
 Timesharing terminal, 2-1
 TOPS-20 batch system, 1-1
 TOPS-20 commands, 2-4
 EXEC, 2-2
 FILCOM, 2-4
 system, 3-1
 using, 3-18
 \$TOPS20 card, 3-1, 3-16
 \$TOPS20 card format, 3-17
 \$TOPS20 card /switches,
 3-17

User Name, Glossary-5
 User-name,
 \$JOB card, 3-3
 Using cards to enter jobs,
 5-11
 Using EDIT, 2-1
 Using the terminal for
 batch, 5-1
 Using TOPS-20 commands,
 3-18

/WIDTH switch, 3-7, 3-8,
 3-11, 3-17

)

)

)

)

)

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify)_____

Name_____Date_____

Organization_____Telephone_____

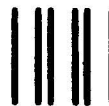
Street_____

City_____State_____Zip Code_____

or
Country

Do Not Tear - Fold Here and Tape

digital



No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

SOFTWARE PUBLICATIONS
200 FOREST STREET MR1-2/E37
MARLBOROUGH, MASSACHUSETTS 01752

Do Not Tear - Fold Here and Tape

Cut Along Dotted Line